Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gregor Stopar

Gradbeni stroji kot senzorji v internetu stvari

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Mojca Ciglarič

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Preglejte področje in tehnologije interneta stvari, osredotočite se zlasti na komunikacije. Raziščite možnosti povezovanja z oblakom. Zasnujte vgrajen sistem na primerni platformi za vgradnjo v gradbene stroje, ki bo spremljal lokacijo in delovanje stroja in pošiljal te podatke v oblak. Sistem implementirajte, preizkusite in komentirajte zasnovo, izvedbo in možnosti uporabe.

Izjava o avtorstvu zaključnega dela

Spodaj podpisani Gregor Stopar, vpisna številka 63090169, avtor zaključnega dela z naslovom:

Gradbeni stroji kot senzorji v internetu stvari (angl. *Construction Machinery as Sensors in Internet of Things*)

IZJAVLJAM

1. da sem pisno zaključno delo študija izdelal samostojno pod mentorstvom doc. dr. Mojce Ciglarič;
2. da je tiskana oblika pisnega zaključnega dela študija istovetna elektronski obliki pisnega zaključnega dela študija;
3. da sem pridobil/-a vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v pisnem zaključnem delu študija in jih v pisnem zaključnem delu študija jasno označil/-a;
4. da sem pri pripravi pisnega zaključnega dela študija ravnal/-a v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil/-a soglasje etične komisije;
5. soglašam, da se elektronska oblika pisnega zaključnega dela študija uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;
6. da na UL neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve avtorskega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja pisnega zaključnega dela študija na voljo javnosti na svetovnem spletu preko Repozitorija UL;
7. dovoljujem objavo svojih osebnih podatkov, ki so navedeni v pisnem zaključnem delu študija in tej izjavi, skupaj z objavo pisnega zaključnega dela študija.

V Ljubljani, dne 2. septembra 2016 Podpis študenta/-ke:

Za vso ponujeno pomoč se zahvaljujem svoji mentorici, doc. dr. Mojca Ciglarič. Zahvaljujem se tudi svoji družini za neizmerno potrpežljivost tekom študija.

Svojim.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	Application Programming interface	aplikacijski programski vmesnik
GPRS	General Packet Radio Service	splošna paketna radijska storitev
GPS	Global Positioning System	globalni sistem za pozicioniranje
IaaS	Infrastructure as a Service	infrastruktura kot storitev
IIoT	Industrial Internet of Things	industrijski internet stvari
IoT	Internet of Things	internet stvari
JSON	JavaScript Object Notation	notacija za JavaScript objekte
M2M	Machine-to-machine	stroj - stroj
OSI	Open Systems Interconnection	osnovni model za komunikacijo
PaaS	Platform as a Service	platforma kot storitev
SaaS	Software as a Service	programska oprema kot storitev
TCP	Transmission Control Protocol	protokol za nadzor prenosa
UDP	User Datagram Protoco	protokol za prenašanje paketov
...	...	...

Povzetek

Naslov: Gradbeni stroji kot senzorji v internetu stvari

V diplomskem delu bomo opravili pregled področja interneta stvari s poudarkom na njegovi uporabi v industriji. Pregledali bomo tudi oblačne storitve in njihovo uporabo v internetu stvari. Implementirali bomo sistem, ki uporablja obe tehnologiji in služi nadzoru nad gradbeno mehanizacijo. Predstavljene bodo izbrane komponente, tehnologije in protokoli ter zasnova sistema. Izdelali bomo vgrajeni sistem, s katerim bomo preko senzorjev pridobivali podatke iz strojev in jih pošiljali v oblačno aplikacijo, prek katere bo uporabnik dostopal do obdelanih podatkov. Podrobneje bomo predstavili različne vrste komunikacij, ki potekajo v sistemu. V zaključku bomo predstavili možnosti za nadaljnji razvoj projekta in podali stroške naše rešitve.

Ključne besede: internet stvari, vgrajeni sistem, oblačna storitev, gradbeni stroji.

Abstract

Title: Construction Machinery as Sensors in Internet of Things

In diploma thesis we will make an overview of the field of internet of things with a focus on its use in industry. We will overview cloud services aswell and its integration with internet of things. We will implement a system which uses both technologies and serves as a control system for construction machinery, where we will present chosen components, technologies, protocols and design of system. We will create an embedded system with sensors for gathering information from machines and sending them to cloud application where user will be able to access processed information. All the communications in the system will be thoroughly presented. In conclusion we will show future options for project development and calculate all costs of the solution.

Keywords: internet of things, embedded system, cloud service, construction machines.

Poglavje 1

Uvod

Trenutni tehnološki razvoj gre v smeri izdelave pametnih naprav, pri čemer ne gre zgolj za proizvodnjo novih pametnih naprav, temveč tudi za nadgradnjo obstoječih. Pod konceptom interneta stvari (ang. Internet of Things – IoT) se tako predvideva, da bo do leta 2020 takih naprav več deset milijard, ki bodo med seboj komunicirale in si izmenjevale informacije. Tak razvoj izmenjave informacij je seveda še posebej dobrodošel v industriji.
Ideja za diplomsko nalogo je nastala pri opazovanju poslovanja skupine RENTING, ki se ukvarja z izposojo gradbenih strojev. Podjetje za večje gradbene stroje ponudi ceno stroja na dan, pri čemer je pogodba velikokrat sestavljena tako, da je delovni dan predviden kot 8 efektivnih delovnih ur. Stranke seveda ure lahko presežejo, a se te obračunajo dodatno. Prvi izziv je torej obračun izposoje. Zaželjeno bi bilo tudi sledenje lokaciji stroja. Ker podjetje stroje izposoja tudi na odročnejše lokacije, na primer v drugo državo, bi se s tem lahko znebili plačevanja zavarovanja proti kraji, ki je za podjetje z veliko stroji precej velik strošek. Pomembno bi bilo vedeti tudi, v kakšnih pogojih se nahaja stroj, na primer temperatura in vlažnost okolja ter pri daljših najemih preverjanje, ali je za stroj potrebno narediti servis. Tak sistem za sledenje strojev ne bi bil primeren samo za izposojevalce gradbenih strojev, temveč tudi za gradbena podjetja, ki bi želela imeti boljši pregled nad svojo mehanizacijo.

Cilj diplomskega dela je izdelava vgrajenih sistemov v strojih in aplikacije, prek katere zbiramo podatke o lokaciji, delovanju in okolju stroja, do katerih dostopamo prek storitve v oblaku, kjer ima uporabnik na voljo grafični vmesnik za dostop do želenih informacij. Podatki se v oblaku hranijo v relacijski podatkovni bazi.

V prvem delu bomo najprej naredili pregled področja same gradbene mehanizacije in koncepta interneta stvari ter opredelili njegovo vlogo in aplikacije v industriji. Pregledali bomo tudi področji vgrajenih sistemov in računalništva v oblaku ter njuno uporabo znotraj IoT.

V drugem delu bomo naredili pregled vseh komponent, ki bodo sestavljale naš sistem, in za vsako opredelili vlogo, ki jo opravlja v sistemu.

Poglavje 2

Pregled področja

2.1 Internet stvari

Internet stvari je pojem, ki je bil prvič uporabljen leta 1999 in se je nanašal predvsem na entitete, ki so bile med seboj povezane prek Radio-frequency identification (RFID) čipov. Danes je IoT mnogo več kot to in sedaj imamo bolj splošno definicijo, ki pravi, da gre za sistem, v katerem so med seboj na različne načine povezane računske, mehanske in digitalne naprave, objekti, živali ali ljudje, ki imajo unikatne identifikatorje in so povezani prek omrežja, predvsem z namenom machine-to-machine (M2M) komuniciranja. Pomembno je tudi, da pri tem ne potrebujejo neposredne človeške interakcije [1]. „Stvar“ v internetu stvari torej predstavlja končno točko v sistemu, ki vsebuje senzorje in druge pripomočke za zbiranje podatkov ter velikokrat nek vgrajeni sistem, ki je tudi sam sposoben procesiranja podatkov, te pa nato po omrežju pošilja za nadaljnjo obdelavo. Leta 2015 smo po raziskavi podjetja Juniper Research znotraj interneta stvari našteli 13,4 milijarde naprav, do leta 2020 pa se predvideva da bo takih naprav kar 38,5 milijarde, kar bi pomenilo porast za več kot 285 % [2]. Glede na te številke si lahko predstavljamo, da se bo IoT razširil na vsa področja naših življenj. Raziskava podjetja IoT Analytics iz februarja 2015, ki je priljubljenost merila glede na objave na Twitterju in LinkedIn-u, je objavila 10 trenutno najbolj

popularnih področij [3]:

1. Pametna hiša (Smart home)
2. Pametni dodatki (Smart wearables)
3. Pametno mesto (Smart city)
4. Pametno električno omrežje (Smart grid)
5. Industrija (Industrial internet)
6. Omrežje avtomobilov
7. E-zdravstvo (Connected health)
8. Pametno nakupovanje (Smart retail)
9. Pametna nabavna veriga (Smart supply chain)
10. Pametno kmetovanje (Smart farming)

Raziskava Juniper Research in IoT Analytics med drugim še ugotavlja, da področja IoT, ki ciljajo na končne uporabnike, kot so Smart home, Smart wearables in podobna, zaenkrat predstavljajo tržni segment uporabnikov z nadpovprečnimi prihodki. Za industrijska področja je bilo medtem ugotovljeno, da gre za področja, ki imajo med omenjenimi daleč največji potencial, a še niso uspela narediti takega preboja kot na primer pametna hiša in pametni dodatki. Raziskava izpostavlja tudi, da IoT projekti v industriji prinašajo zelo velik ROI (return on investment oziroma donos glede na investicijo). Nekaj takih primerov si bomo pogledali v spodnjem podpoglavju.

2.1.1 Uporaba interneta stvari v industriji

Ko govorimo o IoT v industriji, se v tem kontekstu velikokrat uporablja izraz IIoT – Industrial Internet of Things. Ta zajema vključevanje sredstev poslovnega subjekta z namenom lažjega nadzora, pospeševanja proizvodnih

procesov in olajševanja storitvenih dejavnosti podjetij. Izpopolnjena uporaba IIoT v industriji bi v prihodnje lahko prinesla novo industrijsko revolucijo. Ena izmed takih vizij je Industrie 4.0, ki se sicer nanaša zgolj na proizvodne procese. Industrie 4.0 je visokotehnološki projekt nemške vlade, ki predvideva popolno avtomatizacijo proizvodnje, in sicer z nadgradnjo proizvodne opreme do te mere, da bi se ta skozi M2M komunikacijo sama odzivala na morebitne okvare, izvajala vzdrževalna dela in tako bila popolnoma avtonomna. A kot smo že omenili, IoT v industriji ni omejen zgolj na proizvodnjo. Kot primer odlične prakse vključevanja IoT v poslovni proces naj omenimo primer podjetja Deere & co., vodilnega svetovnega proizvajalca kmetijske in gradbene mehanizacije. Podjetje je razvilo sistem MyJohnDeere, ki uporabnikom njihove mehanizacije omogoča naslednje:

- trenutni in pretekli GPS položaj mehanizacije. Uporabnik lahko tako ob večjem številu strojev in delavcev popolnoma nadzira opravljeno delo,
- opozorila za kakršnokoli nenavadno obnašanje stroja, preseženo število dnevnih delovnih ur in opozorila pred morebitnimi okvarami,
- pošiljanje podatkov iz nadzorne plošče (računalnik, tablični računalnik ali telefon) v vse stroje, s čimer lahko na primer pošiljamo načrt dela in druge informacije,
- neposredno povezavo iz strojev do podpornega centra podjetja Deere & Co., kar omogoča pooblaščenemu serviserju vpogled v stanje stroja. Iz oddaljenosti lahko tako uporabniku pomaga prilagoditi nastavitve in preveriti stanje stroja,
- preverjanje kvalitete opravljenega dela, ki se določi s pomočjo senzorjev,
- shranjevanje (backup) podatkov,
- opozarjanje o vremenskih napovedih za izbrano območje,
- in še več.

Podjetje Deere Co. tako svojim strankam nudi dodano vrednost, saj stranke na ta način delo opravijo bolj kakovostno, hitreje in z manj izgube časa zaradi nedelovanja mehanizacije. Po drugi strani podjetje na ta način lažje trži svoje izdelke in nakazuje smernice, proti katerim se bodo gibali trendi v kmetijski panogi [4].

2.2 Računalništvo v oblaku

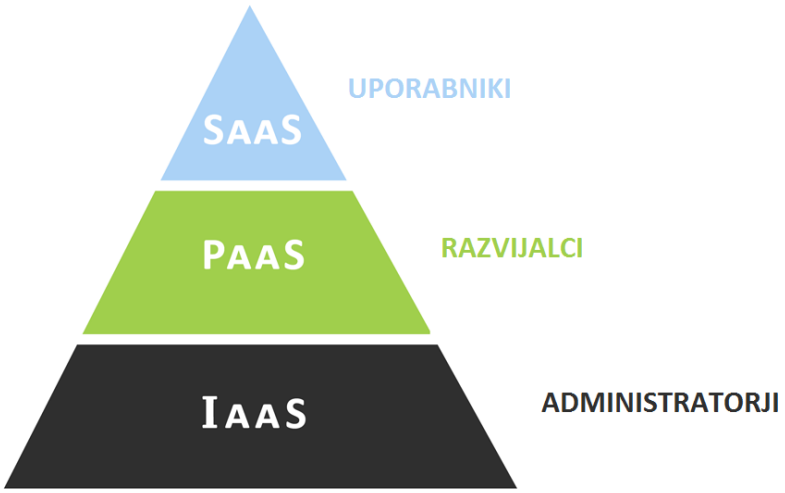
Termin računalništvo v oblaku (ang. cloud computing) je v računalništvu en izmed najpogosteje uporabljenih v zadnjem času. Gre za koncept deljenega izkoriščanja računskih virov, ki niso dostopni lokalno, ampak do njih dostopamo prek interneta. Prav tako imamo tudi podatke shranjene v internetu, in sicer v velikih podatkovnih centrih (ang. data centers). Danes, 9 od 10 podjetij uporablja vsaj eno oblačno storitev [5].

2.2.1 Prednosti računalništva v oblaku

Kljub temu, da je lokalni dostop enostavnejši in hitrejši, ima računalništvo v oblaku nekaj bistvenih prednosti:

- dostop do podatkov in virov kjerkoli in kadarkoli imamo internetno povezavo,
- razširljivost strojne opreme. S serverjem, ki ga imamo doma ali v podjetju, smo omejeni na njegovo kapaciteto, medtem ko imamo pri oblaku zmožnost razširljivosti kapacitet glede na naše potrebe,
- obračun glede na porabo. Stroški uporabe oblaka so veliko bolj fleksibilni, saj veliko ponudnikov nudi obračun glede na dejansko porabo. Nasprotno je, ko sami nabavljamo strojno opremo, saj vedno kupujemo kapacitete glede na predvideno zgornjo mejo uporabe, kar pa v praksi pomeni, da je veliko kapacitet neizkoriščenih,

- hkratni dostop več oseb hkrati, kar izboljša produktivnost skupinskega dela,
- večja varnost, saj pri izgubljenem prenosnem računalniku izgubimo tudi vse podatke, medtem ko v oblaku ostanejo,
- dostop do zadnje verzije programske opreme. V oblaku je programska oprema praviloma vedno nadgrajena na zadnjo verzijo.



Slika 2.1: Naslovniki posameznega storitvenega modela.

2.2.2 Storitveni modeli računalništva v oblaku

Računalništvo v oblaku nam ponuja storitve. Pri tem ločimo tri osnovne storitvene modele, in sicer infrastrukturo kot storitev (ang. Infrastructure as a Service – IaaS), platformo kot storitev (ang. Platform as a Service – PaaS) in programsko opremo kot storitev (ang. Software as a Service – Saas).

Infrastruktura kot storitev

Infrastruktura kot storitev (IaaS) je ponudba računskih virov, podatkovnega prostora in druge IT infrastrukture tipično namenjena sistemskim administratorjem, omrežnim arhitektom ali pa ponudnikom gostovanja oblačnih platform. Nekatere izmed IaaS storitev so virtualni stroji, serverji, omrežja (VLAN), gruče itd. Najbolj znani IaaS produkti so Amazon Web Services (AWS), Microsoft Azure, Cisco Metapod, Google Compute Engine (GCE).

Platforma kot storitev

Platforma kot storitev (PaaS) je storitev, ki se od IaaS ločuje po tem, da vsebuje nameščen operacijski sistem (ang. operating system - OS), izvajalno

Slika 2.2: Primerjave med storitvenimi modeli oblačnih storitev

okolje in včasih programsko opremo za določene posebne storitve, ki jih ne nudi OS (ang. middleware - MW). Namenjena je razvijalcem programske opreme, ki želijo namestiti svojo storitev in jo nato nuditi končnim uporabnikom kot SaaS. Taki primeri so recimo Red Hat Openshift Public PaaS, Google App Engine, Apprenda PaaS.

Programska oprema kot storitev

Programska oprema kot storitev (SaaS) je storitev najvišjega nivoja računalništva v oblaku in je namenjena izključno končnim uporabnikom, ki želijo uporabljati storitev. Gre za najbolj poznane storitve, to so email, CRM, ERP, urejevalniki besedil v oblaku in podobno. Med najbolj poznane produkte spadajo Dropbox, Gmail, Google Apps.

2.2.3 Postavitveni modeli računalništva v oblaku

Poleg storitvenih modelov ločimo tudi postavitvene modele, ki se med seboj razlikujejo po tem, kdo vse lahko dostopa do storitev. To je potrebno predvsem zaradi varnosti, saj mnogo podjetij v oblaku uporablja občutljiv

tip informacij. Ločimo štiri primarne postavitvene modele oblakov in sicer javni, privatni, hibridni in skupnostni oblak [6].

Javni oblak

Javni oblak predstavlja " pravo“ oblačno postavitev z vsemi prednostmi, zaradi katerih se je začelo razmišljati o računalništvu v oblaku. V tem modelu so storitve ponujene raznim strankam, ki jih koristijo zastonj ali po načinu " plačaj, kolikor porabiš“. Primer javne oblačne storitve je iskalnik Google, kjer lahko vsak zastonj uporablja iskalnik. Javni oblak je najbolj primeren za probleme, kjer je potrebna velika skalabilnost in kjer se poraba storitve zelo razlikuje. Tak primer je spletna trgovina, kjer imamo lahko v času popustov ogromno število odjemalcev, drugi dan pa sredi noči več ur zelo malo ali celo nič. Kot zelo elegantna rešitev se za lastnika spletne trgovine tukaj izkaže ravno javni oblak po principu " plačaj, kolikor porabiš“, saj bi bil nakup primerne lastne infrastrukture zgrešen v vsakem primeru. Če bi kupili dovolj zmogljivo in precej drago, ki bi zadostovala konicam obremenjenosti, bi bilo potratno, ker bi bila veliko časa neizkoriščena, v nasprotnem primeru, ko bi kupili manj zmogljivo, bi izgubili poslovno priložnost, saj bi ob veliki obiskanosti trgovina odpovedala.

Privatni oblak

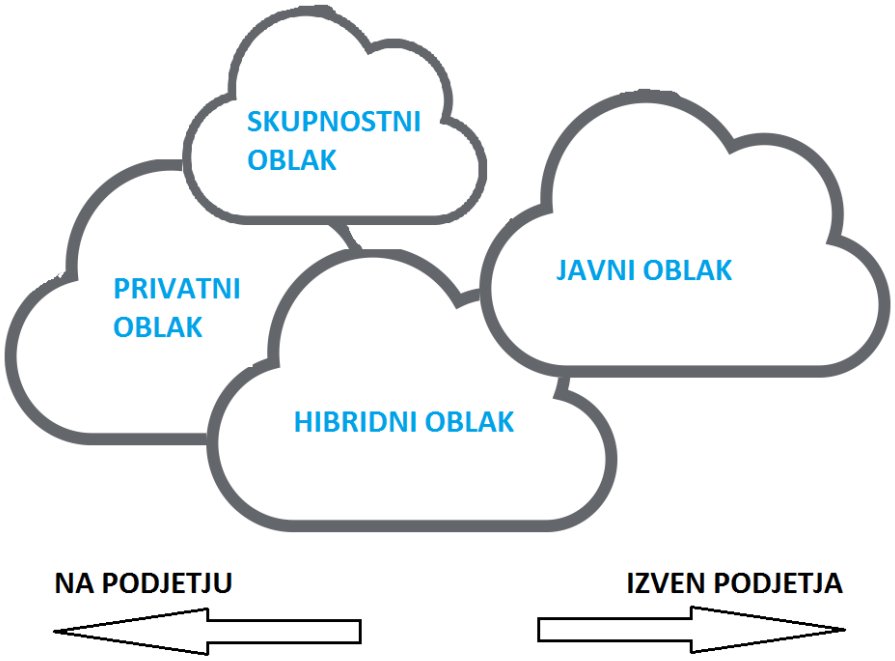
Privatni oblak je bila prva rešitev, ki se je pojavila zaradi pomislekov glede varnosti podatkov s strani organizacij, ko so te prvič prešle na oblačne storitve. Kot varnostna rešitev se sicer obnese zelo dobro, a je v osnovi glede stroškovnih učinkovitosti podobno, kot če bi kupili, postavili in vzdrževali lastno infrastrukturo. Privatni oblak je lahko postavljen na mestu organizacije (ang. on-premise) same ali pri zunanjem ponudniku, čemur rečemo tudi virtualni privatni oblak [6], kakršnega ponuja na primer tudi Amazon. Primeri SaaS aplikacij, ki so velikokrat v obliki privatne oblačne postavitve, so programi za upravljanje odnosov s strankami (ang. customer relationship managment – CRM).

Hibridni oblak

Hibridni model oblaka združuje tako privatni oblak, ki je nastanjen na sedežu organizacije (on-premise), ter javni oblak, ki ga dobavlja ponudnik oblačnih storitev. Tako lahko varnostno občutljive podatke hranimo na privatnem oblaku, ostalo pa na javnem, kar nam omogoča, da še vedno uživamo skalabilnost javnega oblaka. Pri hibridnem oblaku nam lahko javni služi tudi kot neka dodatna infrastruktura, ki jo uporabimo, ko privatni oblak ni zmožen obdelati določene obremenitve. Primer odprtokodne rešitve, ki lahko na tak način poveže privatni in javni oblak, je Eucalyptus [7], ki ima dogovor z Amazon Web Services, da lahko svoj privatni oblak poveže z Amazon Elastic Compute Cloud (EC2), ki je na voljo za uporabo ob povečanih obremenitvah. Skupaj tako tvorita hibridni oblak. Veliko rešitev je tudi takih, kjer imamo PaaS s posebnimi API-ji za integracijo z aplikacijami, ki jih gostimo na privatnem oblaku [6].

Skupnostni oblak

Skupnostni oblak (ang. Community cloud) se nanaša na oblačne storitve, do katerih ima dostop končna množica organizacij ali zaposlenih (npr. v velikih bankah) [8]. Člani skupnosti v skupnostnem oblaku imajo po navadi podobne varnostne, zasebnostne in učinkovitostne zahteve. Tak primer bi bil testiranje varnostno kritičnih produktov, ki jih želimo pozneje prenesti na javni oblak [9]. Vladne organizacije ali tudi privatna podjetja z visoko regulacijsko zakonodajo bi na takem oblaku testirala svoje izdelke najprej znotraj skupnosti. Drugi primer bi bil, da imamo več organizacij, ki želi uporabljati isto aplikacijo. Namesto, da aplikacijo postavimo na vsak server za vsako organizacijo posebej, imamo lahko eno instanco aplikacije in nato na logičnem nivoju razvrstimo posamezne uporabnike. Slednji tako uporabljajo iste kose strojne opreme za dostop do iste aplikacije, kar storitev dela skupnostno [9].

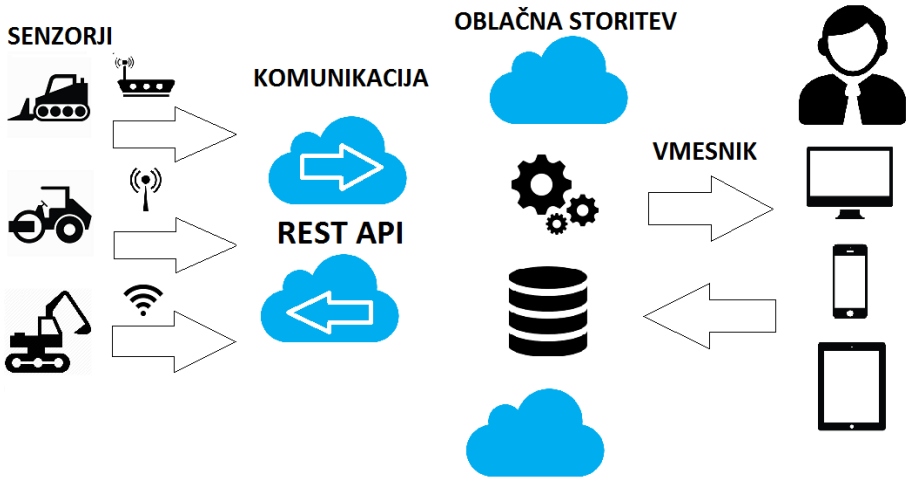


Slika 2.3: Primerjave med postavitvenimi modeli oblačnih storitev

2.2.4 Računalništvo v oblaku in internet stvari

Kot smo že ugotovili v prejšnjih poglavjih, je internet stvari ohlapno definiran kot tehnologija, prek katere komunicirajo različne entitete, ki so sposobne zbiranja in procesiranja podatkov. Če hočemo, da je to uporabno, je potrebno vključiti tudi človeka, v nekaterih primerih zgolj zaradi nadzora, na primer pri proizvodnji, kjer je komunikacija med stroji razvita do te mere, da lahko sami izvajajo potrebna opravila, v drugih primerih pa zato, ker človek iz sistema dobiva podatke za obdelavo, analizo in podobno. Primer slednjega je bil v enem izmed zgornjih poglavij omenjen sistem MyJohnDeere, kjer uporabnik pridobiva koristne informacije iz strojev. Preko računalništva v oblaku lahko te informacije uporabniku podamo kot storitev. Tukaj najdemo prepletanje med omenjenima tehnologijama. Računalništvo v oblaku s sposobnostjo elegantnega reševanja problema vedno večjega števila podatkov in vedno večje potrebe po strojni opremi predstavlja idealno rešitev za nekakšen »front-end« v internetu stvari. Če upoštevamo še, da so oblačne storitve varne in dostopne kjerkoli in kadarkoli, lahko vidimo, da je že sedanjost, sploh pa prihodnost interneta stvari prav v tesnem sodelovanju z računalništvom v oblaku. Na trgu je že precej rešitev, ki kažejo neposredno v to smer. Internet of Things solutions on Google Cloud Platform, Oracle Internet of Things Cloud Service in Salesforce IoT Cloud je samo nekaj rešitev izmed " velikih", vedno več pa je tudi Startup-ov, ki poizkušajo narediti integracijo med tehnologijama, kolikor se da poenostavljeno. Vse rešitve ponujajo podobno, in sicer:

- podporo za programiranje mikrokrmilnikov,
- oblačno storitev tipa PaaS,
- podporo za programiranje spletne aplikacije in podatkovno bazo,
- podporo za izdelavo API-jev za pridobivanje podatkov iz naprav.



Slika 2.4: Oblačne storitve v internetu stvari

2.3 Vgrajeni sistemi

Vgrajeni sistem (ang. Embedded system) je specializiran računalniški sistem, ki je del večjega sistema ali stroja [10]. Glavni gradniki so napajanje, procesor, spomin, časovniki, vrata za serijsko komunikacijo in V/I komponent. Tipično je vgrajeni sistem na mikroprocesorski plošči (98 % mikroprocesorjev danes je komponent vgrajenih sistemov) in je zelo omejen s spominom (tipa ROM). Nekateri vsebujejo operacijski sistem (RTOS – real time operating system), veliko pa je takih, ki so ozko specializirani in vsebujejo zgolj en program. Gre torej za zelo majhne komponente, ki večinoma opravljajo primitivne naloge, kot so na primer krmiljenje pritiskov na gumbe, prikaz informacij in podobno, najdemo pa tudi take ki opravljajo kompleksnejše naloge, kot je odzivanje na neke dogodke. Tak primer je tudi vgrajeni sistem za obveščanje o vremenu, ki je sprogramiran, da oddaja informacije na 30 minut, a ima vseeno tudi logiko za odziv na nenadno spremembo in takojšnje pošiljanje o tem.

2.3.1 Vgrajeni sistemi znotraj interneta stvari

Glede na zgornji opis si lahko predstavljamo, da so vgrajeni sistemi v IoT prisotni v vseh končnih točkah (t. i. „stvareh“). Prav vgrajeni sistemi so namreč tisti, ki omogočajo, da lahko tako širok nabor predmetov, ljudi in živali predstavlja „stvar“ v IoT. Za enkrat sta pri večini vgrajenih sistemov v IoT primarni nalogi pridobivanje podatkov iz senzorjev in pošiljanje teh naprej po omrežju za nadaljnjo obdelavo, torej brez logike za kompleksnejše procesiranje. A v nekaterih primerih, sploh pa v prihodnosti, naj ne bi bilo več tako. Predvsem v industriji se že pojavlja višje nivojski vgrajeni sistem imenovan kiber-fizični sistem (CPS – cyber physical system), ki je sposoben kompleksnejše kombinacije in koordinacije med fizičnimi in računskimi elementi. CPS je implementiran tako, da vsako fizično „stvar“ dopolnjuje dvojček v obliki kiber procesa, ki je po navadi v oblaku [11]. Z medsebojno povezavo več CPS lahko v primerjavi s sedaj znanimi rešitvami na področju avtomatizacije dosežemo sistem z višjo stopnjo avtonomnosti, prilagodljivosti, učinkovitosti, varnosti, funkcionalnosti, zanesljivosti in predvsem uporabnosti.

2.3.2 Izvedba vgrajenega sistema

V zadnjem času se je na področju vgrajenih sistemov zelo razvil t. i. „Do it yourself“ oziroma DIY koncept, ki ga je omogočil pojav razvojnih plošč za izvedbo vgrajenih sistemov. Pogoj za take plošče je, da so razmeroma enostavne za uporabo in cenovno dostopne ter temeljijo na odprtem sistemu in odprto kodnih knjižnicah, kar jih dela primerne za razvoj raznih prototipov na področju vgrajenih sistemov. Najbolj poznane in uporabljane razvojne plošče so plošče podjetij Arduino in Raspberry Pi, za katera imamo na voljo ogromno t. i. modulov, med katerimi najdemo module za komunikacijo prek različnih tipov omrežij, senzorje, kot so na primer GPS senzorji, senzorji za temperaturo in vlažnost, kamere, ekrane in še veliko več. Kljub podobnim možnostim, ki nam jih nudijo plošče obeh proizvajalcev, pa je med njimi

precej bistvenih razlik. Arduino plošče so plošče z mikrokontrolerjem oziroma zelo enostavnim računalnikom, ki je sposoben poganjati en program na enkrat. Medtem ko so plošče Raspberry Pi polno opravilni računalniki s podobnimi komponentami, kot jih imajo PC-ji, na katerih po navadi teče Linux operacijski sistem in imajo zmožnost poganjati več opravil hkrati.

2.4 Komunikacija v internetu stvari

Komunikacija v internetu stvari je seveda zelo pomembna, saj če želimo da neka „stvar“ dejansko postane del interneta stvari, mora biti ta povezana v internet in komunicirati z ostalimi. Komunikacija je izvedena po TCP/IP modelu, ki vsebuje štiri plasti [12]:

- **plast omrežnega vmesnika** (ang. network interface layer), ki glede na OSI model združuje fizično in povezovalno plast.
- **internetno plast** (ang. internet layer) oziroma, kot jo Cisco imenuje, gostitelj – gostitelj plast [12],
- **transportno plast** (ang. transport layer),
- **aplikacijsko plast** (ang. application layer), ki glede na OSI model sedmih plasti združuje zgornje tri, in sicer aplikacijsko, predstavitveno in sejno.

Različnih implementacij na posameznih plasteh je več, te pa so odvisne predvsem od lastnosti naprav, ki jih povezujemo.

2.4.1 Plast omrežnega vmesnika

Na plasti omrežnega vmesnika največkrat srečamo naslednje protokole [14]:

- Ethernet (10, 100, 1G)
- WiFi (802.11b, g, n)

- Serijsko s PPP
- GSM, 3G, LTE, 4G

Med naštetimi so vedno bolj priljubljene tehnologije mobilnega interneta, saj nam omogočajo internetno povezavo naprav tudi z oddaljenih lokacij. Potencial, ki ga internet stvari s svojim obsegom za ponudnike mobilnih omrežij prinaša, je zares velik. To so spoznali tudi večji ponudniki, ki že ponujajo posebne pakete za naprave v internetu stvari.

2.4.2 Internetna plast

Na internetni plasti je za prenos paketov preko omrežja na večini naprav še vedno uporabljen protokol IPv4, ki pa ga bo zaradi zapolnjenosti naslovov v prihodnosti v celoti zamenjal protokol IPv6. Bloki IPv4 naslovov, ki jih dodeljuje inštitut IANA (Internet Assigned Numbers Authority), so pošli že 31. 1. 2011. Poleg IPv6 moramo tukaj omeniti še protokol 6LoWPAN (IPv6 over Low Power Wireless Personal Area Networks), ki je, kot že ime pove, namenjen napravam z malo energije in omejenimi procesnimi sposobnostmi, z namenom, da so lahko tudi te del interneta stvari.

2.4.3 Transportna plast

Na transportni plasti je najbolj uporabljan TCP protokol, a v prihodnje naj bi bila pogostejša uporaba protokola UDP. Težava pri TCP protokolu je potratnost energije, predvsem pri vgrajenih sistemih, ki so baterijsko napajani, ter da protokol ni popolnoma realno časoven in deterministično [15]. Problem je potrjevanje paketov, kjer mora pošiljatelj (po navadi naprava z malo energije) po poslanem paketu čakati na potrditev. UDP je tudi bolj primeren v podomrežju, kjer imamo ogromno število naprav, saj UDP prenosi zahtevajo zgolj dva UDP datagrama, za vsako smer enega. Na ta način je obremenjenost omrežja zelo zmanjšana.

2.4.4 Aplikacijska plast

Najbolj uporabljani protokoli aplikacijske plasti so RESTful HTTP, WebSocket, CoAP, XMPP in MQTT. Razlikujejo se v uporabi protokola na transportni plasti, pri čemer se uporabljata TCP in UDP, in po načinu sporočanja, kjer je uporabljen model Request/Response ali Publish/Subscribe. Pri prvem gre za način sporočanja, kjer dve napravi komunicirata prek zahtevkov in odgovorov neposredno ena z drugo in sprejemata vse, kar druga naprava pošlje. Pri modelu Publish/Subscribe je sporočanje posredno preko t. i. „brokerja“. Naprava, ki želi nekaj poslati, najprej pošlje „brokerju“ („publish“ del komunikacije). Druga naprava, ki želi sprejeti, kar je naprava poslala brokerju, se na tega prijavi („subscribe“ del komunikacije) in tako sprejme sporočila. Medtem je lahko pošiljatelj poslal tudi več kot eno sporočilo. V Tabeli 2.1 so prikazane še dodatne primerjave med protokoli.

RESTful HTTP

RESTful HTTP protokol je najbolj poznan, hkrati temelj modela odjemalec – strežnik in je pogosto uporabljan tudi v internetu stvari. Velikokrat je zaradi varnosti implementacija izvedena tako, da imamo na napravi samo odjemalca, saj je varneje, da lahko naprava samo začne povezavo, ne pa tudi sprejema. Primeren je za uporabo predvsem, ko želimo imeti zagotovljen prenos podatkov, saj gre na transportni plasti prek TCP protokola. Način sporočanja je Request/Response, kjer prejemnik vedno odgovori na zahtevo.

WebSocket

WebSocket je protokol, ki prek ene TCP povezave omogoča full-duplex (dvosmerno) komunikacijo. Na ta način je dvosmerna komunikacija zelo poenostavljena, a praviloma je implementirana samo tam, kjer je to res potrebno (na primer, ko želimo neko IoT napravo krmiliti prek interneta).

XMPP

XMPP (Extensible Messaging and Presence Protocol) je protokol, ki temelji na XML strukturi pošiljanja podatkov in je namenjen realno-časovni komunikaciji. Izmenjava sporočil je izvedena po Publish/Subscribe principu, na transportni plasti pa je uporabljen TCP protokol.

CoAP

CoAP (Constrained Application Protocol) je protokol, ki je bil narejen posebej za enostavne elektronske naprave. Je zlahka prevedljiv v HTTP za spletno komunikacijo, primeren pa je za internet stvari predvsem zato, ker je zelo enostaven, z majhnim *overheadom* in možnostjo *multicast* pošiljanja paketov. Za razliko od HTTP-ja je utemeljen na UDP protokolu na transportni plasti, način sporočanja pa je Request/Response.

MQTT

MQTT (MQ Telemetry Transport) je protokol, ki je bil narejen posebej za M2M komunikacijo in internet stvari. Je zelo „lahek“, kar pomeni, da je primeren za omrežja z majhno pasovno širino in velikimi latencami. Protokol je uporaben v zelo velikih omrežjih, kjer imamo majhne naprave, ki jih želimo kontrolirati in nadzorovati, nimamo pa neposredne komunikacije med napravami. Pošiljanje sporočil je izvedeno po principu Publish/Subscribe, kjer imamo kot posrednika MQTT „brokerja“ za posredovanje sporočil. Na transportni plasti imamo TCP protokol.

Protokol	**RESTful HTTP**	**WebSocket**	**XMPP**	**CoAP**	**MQTT**
Transportnaplast	TCP	TCP	TCP	UDP	TCP
Tip sporočanja	Request/response	Request/response	Publish/subscribe	Request/response	Publish/subscribe
Mobilna omrežja	Zelo primerno	Zelo primerno	Zelo primerno	Zelo primerno	Zelo primerno
Velika omrežja	Primerno	Primerno	Primerno	Zelo primerno	Primerno

Tabela 2.1:

Poglavje 3

Zasnova in izbira komponent, tehnologij ter protokolov sistema

3.1 Zasnova sistema

Od naše rešitve pričakujemo, da bo uporabniku nudila prijazen nadzor nad svojo gradbeno mehanizacijo. V prvi vrsti gre tukaj za GPS sledenje, ki je v osnovi implementirano tako, da stroj periodično oddaja svoj položaj. Poleg položaja stroj oddaja tudi vlažnost in temperaturo okolja ter podatek o tem, ali je stroj v delovanju. Želimo imeti tudi možnost spremembe periode oddajanja in s tem na našem oddajniku prihraniti z energijo. Če želimo, da so tej podatki, kolikor se le da, uporabni, moramo imeti dovršen tudi t. i. „front end“ s prijaznim uporabniškim vmesnikom, hranjenjem pridobljenih podatkov in funkcionalnostmi za njihovo obdelavo. Pričakujemo, da ne bomo imeli samo izpisa GPS koordinat, temveč tudi prikaz strojev na zemljevidu, na katerem bomo lahko označili tudi območja, ki jih bomo lahko dodelili posameznemu stroju. Od sistema bomo pričakovali obveščanja, kdaj je stroj vstopil v dodeljeno območje ali ga zapustil, pa tudi o napakah v GPS podatkih, ki jih sprejemamo, ali če stroj že nekaj časa ni oddal signala.

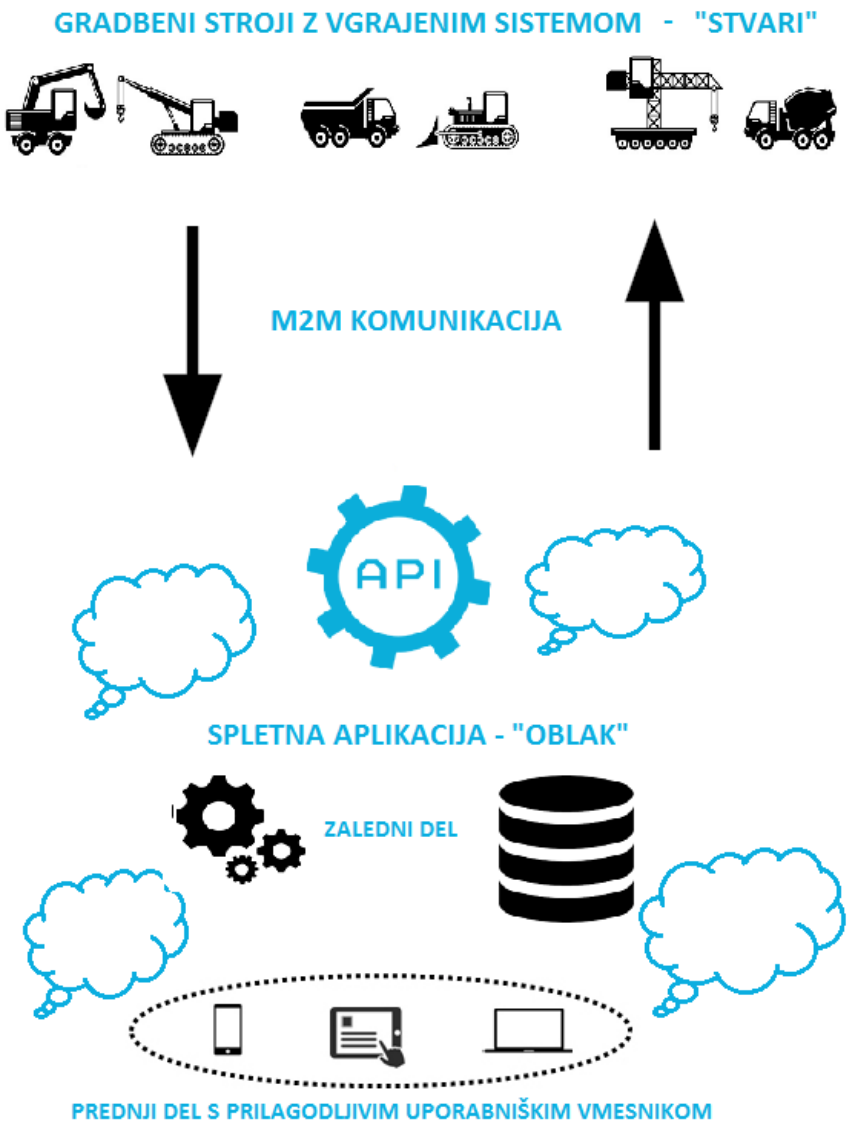
Poleg realno-časovnega GPS sledenja strojev želimo imeti tudi pregled nad zgodovino njihove uporabe. Želimo imeti grafični prikaz uporabe stroja po mesecih in po zadnjih dnevih. Tako lahko točno vidimo, kdaj so bile kakšne omejitve glede ur presežene in kdaj je bila obremenjenost stroja največja, s čimer lahko optimiziramo tudi nabavo rezervnih delov in potrošnega materiala za servisiranje stroja. Pri izposoji stroja želimo v sistemu ustvariti novo izposojo, kjer določimo začetek in predviden konec izposoje. Sistem nas mora ob predvidenem koncu, v kolikor je stroj še v izposoji, opozoriti, da lahko ustrezno ukrepamo. Ob vračilu opreme pričakujemo od sistema, da sam naredi okvirni obračun izposoje, kjer mora poleg osnovnih cen upoštevati tudi prekoračene ure. Od same komunikacije med strojem in našo aplikacijo pričakujemo, da bo ta kar se da energijsko varčna za oddajnik na stroju in časovno ne preveč obsežna za API naše aplikacije. Preko tega želimo namreč prejemati podatke iz večjega števila strojev. Želimo tudi, da, ko povečamo periodo oddajanja podatkov iz stroja, ali ko se signal prekine, sistem na stroju sam hrani začasne podatke o GPS koordinatah, trajanju delovanja stroja in temperaturi ter vlažnosti, dokler se podatki ne odpošljejo.

Zasnovo sistema lahko razdelimo na naslednje dele:

- **„Stvari“**, kar v našem primeru predstavlja gradbeni stroj z vgrajenim sistemom, sestavljen iz krmilnika, GPS senzorja, GSM modula in senzorjev za zaznavanje delovanja stroja in temperature ter vlažnosti okolice,

- **„Oblak“**, aplikacijski del s podatkovno bazo, ki ga delimo na zaledni del, ki skrbi za hranjenje podatkov o uporabniku in strojih ter vsebuje funkcije za njihovo obdelavo in prednji del z uporabniškim vmesnikom, do katerega uporabnik dostopa prek spleta in je zasnovan tako, da je prilagodljiv za delovanje na različnih napravah (računalniku, tabličnem računalniku in telefonu),

- **„M2M komunikacija“**, prek katere pridobivamo podatke iz gradbenih strojev v oblak. Ta mora biti zanesljiva, energijsko varčna in varna.



Slika 3.1: Zasnova sistema.

3.2 Izbrane komponente, tehnologije in protokoli vgrajenega sistema

Kot smo že omenili v prejšnjih poglavjih, želimo, da gradbeni stroji v internetu stvari predstavljajo „stvar“, ki sprejema svoj GPS položaj, zaznava temperaturo in vlažnost, delovanje stroja in nato te podatke pošlje v oblak. Če želimo to doseči, moramo v gradbeni stroj vgraditi vgrajeni sistem, ki vsebuje sledeče:

- **mikrokrmilnik**, s katerim bomo krmilili vse module in senzorje,
- **GPS modul**, ki zaznava GPS položaj stroja,
- **senzor za zaznavanje delovanja stroja**, prek katerega lahko zaznamo 12 V napetost toka, ki se pojavi ob vžigu stroja,
- **senzor za temperaturo in vlažnost**,
- **modul za komunikacijo**, ki služi vzpostavitvi povezave v internet, prek katerega komuniciramo z aplikacijo v oblaku.

V naslednjih podpoglavjih so izbrane komponente in opis implementacije celotnega vgrajenega sistema.

3.2.1 Mikrokrmilna ploščica AT MEGA 2560

Za ploščico z mikrokrmilnikom smo izbrali ploščico MEGA 2560. Ta vsebuje 8-bitni mikrokrmilnik proizvajalca ATMEL ATmega2560. Lastnosti ploščice so predstavljene v Tabeli 3.2.1

Lastnosti ploščice

Mikrokrmilnik	ATmega2560
Delovna napetost	5 V
Vhodna napetost	6-20 V (7-12V priporočeno)
Digitalnih vhodno-izhodnih priklopov	54 (15 z PWM izhodom)
Analognih vhodnih priklopov	16
DC tok na vhodno-izhodni priklop	20 mA
Flash spomin	256 KB (8 KB za bootloader)
SRAM	8 KB
EEPROM	4 KB
Hitrost ure	16 MHz
Dimenzije (dolžina, širina, teža)	101,52 mm, 53,3 mm, 37 g

Tabela 3.1: Tabela specifikacij za ploščico MEGA 2560

3.2.2 GPS ploščica CRIUS NEO-6M

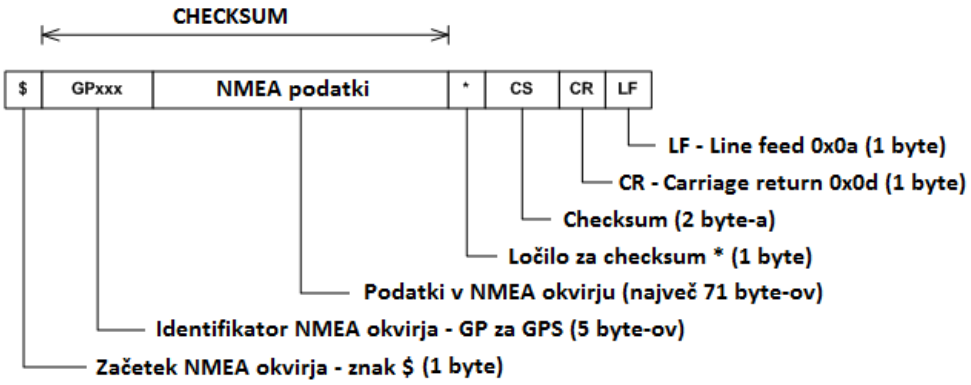
Za zaznavanje GPS položaja je bila izbrana ploščica CRIUS NEO 6M s specifikacijami prikazanimi v Tabeli 3.2 na strani 26.

Tabela 3.2: Tabela specifikacij za CRIUS NEO 6M GPS modul

Lastnosti		
	GPS modul	U-blox NEO-6M
	Frekvenca osveževanja	5 Hz
	Shranjevanje konfiguracije	32k I2C EEPROM
	Vgrajena antena	CIRACOMM 0001 keramična
	Vrata za komunikacijo	UART(TTL)
	Baterija	3V lithium rechargable
	LED lučke	Lučka vklopa in lučka signala
Privzeti parametri		
	Protokol	Standard NMEA
	Hitrost prenosa	9600V
	Hitrost osveževanja navigacije	1 Hz
	Perioda časovnika(LED luči)	1 Hz
Delovni pogoji		
	Temperatura	-40C do 85C
Dodatki		
	Kabel 4x	VCC, RX, TX, GROUND

NMEA protokol

NMEA ali National Marine Electronics Association je neprofitno združenje proizvajalcev, distributerjev, prodajalcev, izobraževalnih ustanov in drugih na področju periferne navtične elektronike, ki izdaja NMEA standarde za komunikacijo med raznimi elektronskimi instrumenti, prisotnimi v navtiki. Naš GPS modul uporablja protokol NMEA 0183, in sicer verzijo 2.3, ki je zamenjal prejšnja NMEA 0180 in NMEA 0182, sprejet pa je bil že tudi NMEA 2000. Protokol določa tipe in format sporočil, ki jih tvorimo z NMEA okvirom prikazanim na Sliki 3.2.

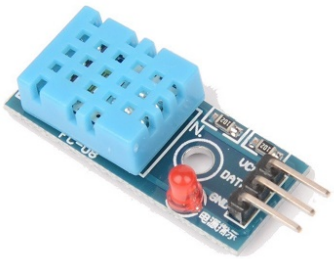


Slika 3.2: NMEA okvir.

Primer NMEA sporočila:

$$\$GPGGA, 123519, 4807.038, N, 01131.000, E, 1, 08, 0.9, 545.4, M, 46.9, M, , *47 \quad (3.1)$$

V prikazu sporočila 3.1 vidimo primer NMEA sporočila tipa GGA, ki nam poda bistvene podatke za določitev 3D položaja in natančnost teh podatkov. GGA je poleg sporočil tipa RMC, ki predstavi osnovne GPS informacije v najmanjši obliki, in GSA, ki nam pove podatke stanja satelitov.



Slika 3.3: DHT11 senzor za merjenje temperature in vlažnosti.

Meritveni razpon	**Natančnost vlažnosti**	**Natančnost temperature**	**Osnovna enota**	**Odzivni čas**
20-90%RH in 0-50°C	+- 5%RH	+- 2°C	1% in 1°C	6s - 30s

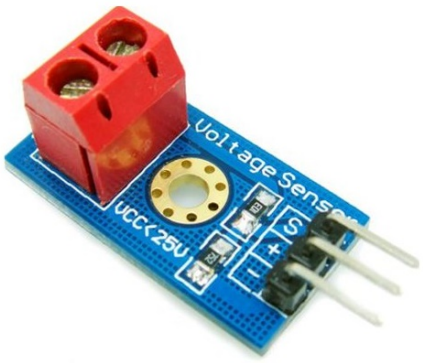
Tabela 3.3: Lastnosti DHT11 senzorja

3.2.3 Senzor za zaznavanje temperature in vlažnosti DAOKAI DHT11

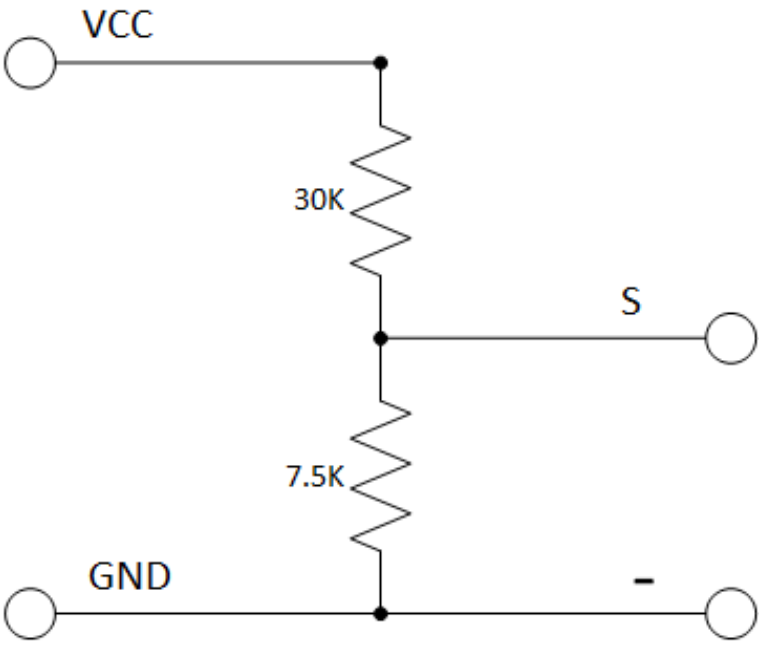
Za zaznavanje temperature in vlažnosti okolice stroja je bil uporabljen senzor DHT 11 na ploščici proizvajalca DAOKAI (Slika 3.4). V Tabeli 3.2.3 so prikazane osnovne lastnosti senzorja.

3.2.4 Senzor za zaznavanje delovanja stroja

Za zaznavanje delovanja stroja je uporabljen senzor za zaznavanje napetosti. Delovanje stroja opazimo tako, da iz stroja zaznavamo napetost električnega kroga, ki se poveča, ko damo kontakt (obrnemo zagonski ključ). Ker je napetost takega kroga več kot 5 V, moramo uporabiti senzor, ki je v resnici napetostni delilnik 5:1, ki uporablja 30 K in 7.5 K Ohm upor (Slika 3.5). Tako lahko napetosti do 25 V prevedemo na napetosti med 0 V in 5 V, ki jih lahko prebiramo iz analognega vhoda na mikrokrmilnik.

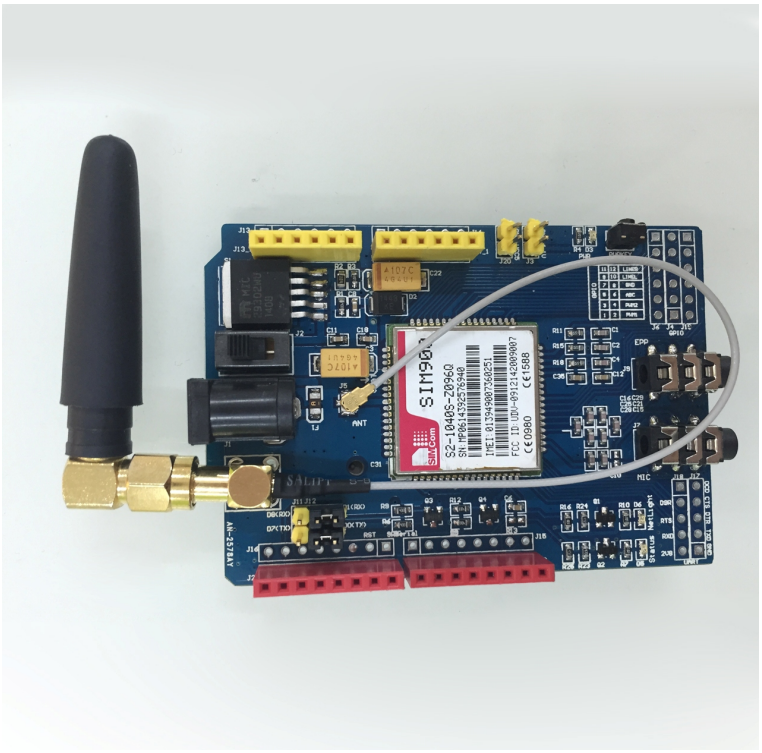


Slika 3.4: Senzor za merjenje napetosti.



Slika 3.5: Shema implementacije senzorja napetosti

3.2.5 Modul za povezavo v internet



Slika 3.6: GPRS shield Arduino UNO

Za povezavo v internet in komunikacijo z oblakom je bila izbrana ploščica GPRS shield Arduino UNO (Slika 3.6). Po zgradbi je v prvi vrsti namenjena mikrokrmilniku Arduino UNO in jo lahko s tem povežemo tako, da jo enostavno namestimo na mikrokrmilnik, saj se pini pozicijsko popolnoma prilegajo v obeh komponentah. Vseeno pa lahko komponento uporabimo tudi pri drugih modelih, in sicer s pravilno vezavo *jumper* žic.
Na ploščici je uporabljen modul SIM900 proizvajalca SIMComm. Je *quad-band* naprava, kar pomeni, da omogoča podporo za vse štiri frekvenčne razpone uporabljene v GSM komunikaciji, in sicer 850 MHz, 900 MHz, 1800 MHz in 1900 MHz. Poleg tega omogoča GPRS povezavo in nudi podporo za TCP/IP sklad ter FTP in HTTP protokol.

3.3 Izbrane komponente, tehnologije in protokoli aplikacije v oblaku

Glede na opis naših pričakovanj, ki jih imamo od rešitve v podpoglavju 3.1, lahko trdimo, da bomo našo rešitev končnemu uporabniku nudili kot oblačno storitev tipa SaaS (glej podpoglavje 2.2.2, kjer so opisani različni storitveni modeli). Kot razvijalci moramo zato izbrati primernega ponudnika storitve PaaS, na kateri bomo postavili našo aplikacijo. V razdelku **Platforma kot storitev** podpoglavja 2.2.2 smo opisali kako izgleda PaaS model, in navedli nekaj primerov storitev različnih ponudnikov. Za naše potrebe je bila izbrana rešitev **Openshift Public PaaS** proizvajalca Red Hat in je podrobneje opisana v podpoglavju 3.3.1. Rešitev je bila izbrana predvsem zato, ker ponujajo možnost brezplačne uporabe omejenih kapacitet, kar je primerno za izvedbo prototipske rešitve. Če smo s ponudbo zadovoljni in želimo razširiti kapacitete, lahko začasni paket enostavno nadgradimo in plačujemo po porabi.
Spletna stran je narejena v tehnologijah HTML5, Javascript in CSS3. Za shranjevanje podatkov je uporabljena relacijska baza MySQL, za strežniški skriptni jezik pa PHP.

3.3.1 OpenShift Public PaaS by Red Hat

Openshift je platforma, ki nam omogoča postavitev aplikacij v več programskih jezikih (Java, Ruby, Node.js, PHP, Python in ostali) ter več tipih podatkovnih baz (MySQL, PostgreSQL, MongoDB in SQLite), do katerih nam omogoča dostop in popoln nadzor. Podporo za tehnologije, ki jih želimo uporabljati na platformi, dodamo kot *cartridge*. Poleg naštetih lahko prek OpenShift Cartridge API-ja tudi sami naložimo druge podatkovne baze, *middleware* ter jezikovne *cartridge-e*. OpenShift nam ponuja tudi uporabo Git-a, programa s pomočjo katerega na strežnik enostavno prenesemo aplikacijo, ki jo razvijamo lokalno.
OpenShift PaaS ponuja kapacitete v obliki koles. Kolo v OpenShift platformi predstavlja skupek kapacitet, sestavljenih iz alocirane centralno pro-

cesne enote – CPE (ang. central process unit – CPU), spomina, diska in pasovne širine.
Na platformi je postavljen strežnik Apache verzija 2.2.15, na katerem je spletna aplikacija. Dostop do aplikacije ima odjemalec prek HTTP protokola na naslovu, ki je sestavljen sledeče:

$$http://xxx-yyy.rhcloud.com$$

Kjer sta:

- xxx - ime aplikacije,
- yyy - naša izbrana domena.

3.3.2 Uporabljene spletne tehnologije odjemalčeve strani

Uporabljene so bile tri osnovne in obenem najbolj razširjene tehnologije za razvoj in prikaz vsebine na WWW (World Wide Web – svetovni splet) in sicer:

- HTML (HyperText Markup Language), verzija 5, označevalni jezik za izdelavo spletnih strani,
- CSS (Cascading Style Sheets), verzija 3, jezik za opis grafičnega prikaza vsebine zapisane v označevalnem jeziku in
- Javascript, objektno-orientiran skriptni jezik za ustvarjanje dinamičnih vsebin.

Za lažjo zagotovitev prilagodljivega uporabniškega vmesnika (ang. responsive design), ki je prilagodljiv za prikaz na računalniškem zaslonu, tabličnem računalniku in telefonu, je bilo uporabljeno ogrodje **Bootstrap**. Gre za najbolj priljubljeno tovrstno ogrodje, ki združuje zgoraj omenjene tehnologije in s katerim dosežemo prilagodljivost uporabniškega vmesnika z uporabo predpripravljenih HTML in CSS elementov. Primer prilagodljivega uporabniškega vmesnika narejenega s pomočjo ogrodja Bootstrap lahko vidimo

Slika 3.7: Primer prilagodljivega uporabniškega vmesnika

na Sliki 3.7.

Uporabljeni sta bili tudi Javascript knjižnica **jQuery**, ki poenostavlja manipuliranje DOM elementov, in tehnologija **AJAX** (Asynchronous JavaScript and XML), ki nam omogoča asinhrone klice strežniških skript.

3.3.3 Spletne tehnologije strežniške strani

Za delo s spletnimi tehnologijami strežniške strani smo na platformi dodali naslednje *cartridge-e* (omenjeni v uvodnem odstavku podpoglavja 3.3.1):

- PHP 5.4
- MySQL 5.5
- phpMyAdmin 4.0
- Cron 1.4

PHP 5.4

Osrednja tehnologija strežniške strani, ki smo jo uporabili v naši aplikaciji, je PHP. Gre za odprto-kodni splošno namenski skriptni jezik, ki je uporaben predvsem za izdelavo dinamičnih spletnih aplikacij [16]. Za razliko od tehnologij odjemalčeve strani se PHP koda izvede na strežniku, kjer tvori HTML

dokument, ki ga nato pošlje odjemalcu, ki vidi zgolj končni izdelek in ne vidi, kaj dejansko je ustvarilo tak dokument. PHP jezik se največkrat uporablja v povezavi z relacijsko bazo MySQL (možna je sicer tudi integracija z Microsoft SQL), z namenom pridobivanja in obdelave shranjenih podatkov.

MySQL 5.5

MySQL je en izmed štirih (poleg Firebird, PostgreSQL in SQLite) vodilnih upravljavskih sistemov relacijskih podatkovnih baz. Tako kot vsi sistemi relacijskih podatkovnih baz tudi MySQL za poizvedovanje in vzdrževanje podatkovne baze uporablja jezik SQL (Structured Query Language).

phpMyAdmin 4.0

Phpmyadmin je orodje, ki nam omogoča administracijo MySQL podatkovnih baz prek spletnega brskalnika [17]. Ponuja nam tudi grafični vmesnik za lažje ustvarjanje, brisanje ali urejanje baz, tabel, polj in vrstic. Omogoča nam upravljanje nad uporabniki baze, kjer lahko uporabnike ustvarjamo in brišemo ter jim dodeljujemo pravice za dostop do baze.

Cron 1.4

Cron 1.4 *cartridge* nam omogoča izvajanje tako imenovanih **„CRON“** opravil. Gre za dodeljevalnik poslov, ki se izvajajo periodično in so uporabni predvsem za čiščenje podatkov, ustvarjanje varnostnih kopij in izdelavo poročil.

Poglavje 4

Implementacija

Implementacijo smo razdelili na naslednje glavne točke:

- implementacija vgrajenega sistema s senzorji v gradbenem stroju,
- implementacija aplikacije v oblaku in
- komunikacija.

4.1 Vgrajeni sistem

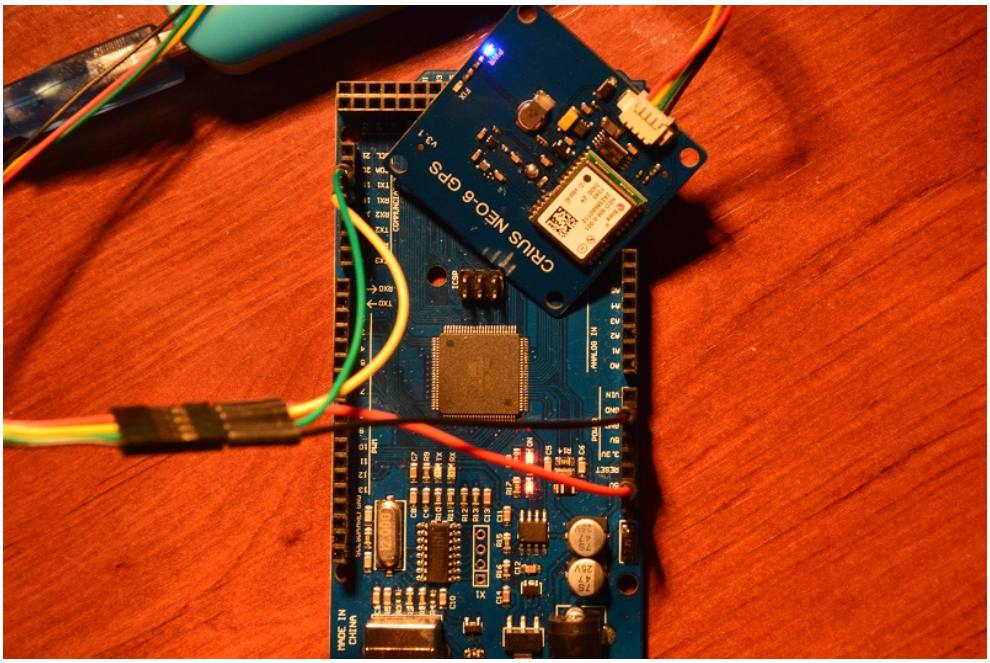
Pri implementaciji vgrajenega sistema bomo predstavili implementacijo mikrokrmilnika MEGA 2560 s posameznimi senzorji, ki smo jih našteli v podpoglavju 3.2. O izvedbi povezave v internet bomo več povedali v podpoglavju 4.3, kjer bomo predstavili celoten koncept komunikacije med vgrajenim sistemom in oblačno aplikacijo.

4.1.1 Pridobivanje GPS podatkov

Povezave med GPS modulom in mikrokrmilnikom

Na Sliki 4.1 je prikazana vezava GPS modula na mikrokrmilnik. Kot vidimo, imamo 4 povezave in sicer:

- z rdečo žico imamo povezavo na 5V električni vir,



Slika 4.1: Povezava GPS modula z mikrokrmilnikom.

- s črno žico imamo povezavo na GROUND oziroma ozemljitev,
- z rumeno žico imamo povezan Tx oziroma oddajni pin na modulu z Rx oziroma s sprejemnim pinom na mikrokrmilniku in
- z zeleno žico imamo povezan Rx oziroma sprejemni pin na modulu s Tx oziroma sprejemnim pinom na mikrokrmilniku.

Programska implementacija

Pri implementaciji smo si pomagali z uporabo knjižnice TinyGPS++. To je odprtokodna knjižnica za Arduino mikrokrmilnike in nam pomaga pri razčlenjevanju NMEA podatkov. Knjižnica je razdeljena na več objektov, prek katerih pridobivamo želene podatke, ne da bi se obremenjevali s tipom in z razčlenjevanjem sporočila. Ti objekti so sledeči:

- **location** – zadnji podatki o lokaciji,

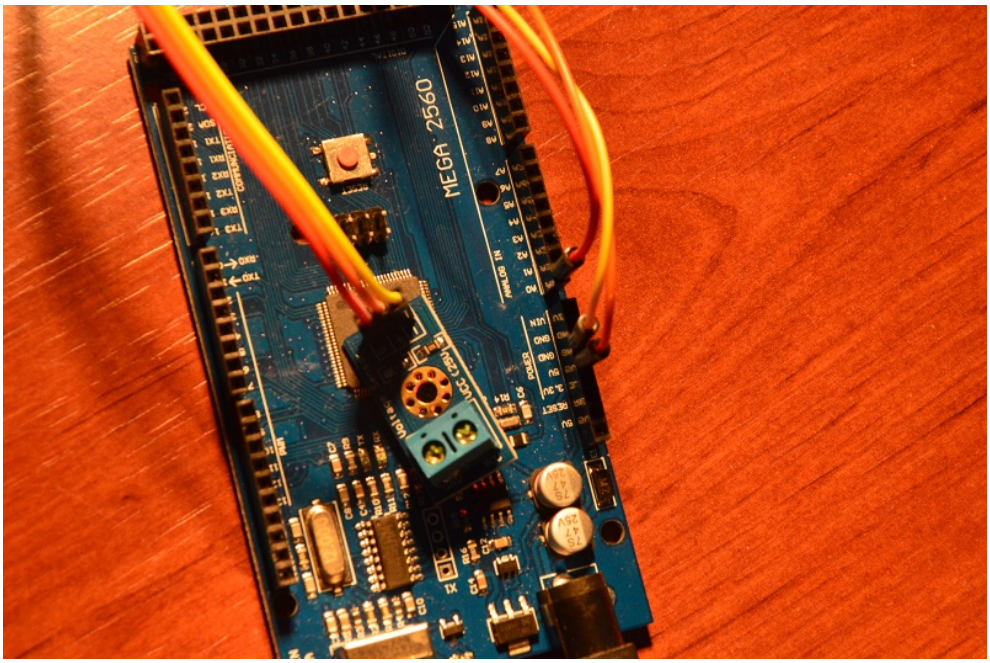
- **date** – zadnji podatek o datumu (UT),
- **time** – zadnji podatek o času (UT),
- **speed** – trenutna hitrost potovanja,
- **course** – trenutni kurz potovanja,
- **altitude** – zadnja zaznana nadmorska višina,
- **satellites** – število vidnih satelitov in
- **hdop** – horizontalni odmik natančnosti.

V funkciji **loop()** prejemamo in uporabljamo podatke iz GPS modula. Če želimo, da TinyGPS++ dela pravilno, moramo neprestano prebirati podatke iz GPS modula s pomočjo **encode()** funkcije. To je pomembno predvsem zaradi tega, ker prihajajo iz GPS modula različni tipi sporočil. Če na primer preberemo samo eno sporočilo, ga v zanki obdelamo in nato ponovno preberemo samo eno sporočilo ter gremo spet v zanko, se nam lahko zgodi, da bomo imeli v drugem obhodu zanke nekatere podatke enake kot v prvem obhodu, saj prebrano sporočilo določenih podatkov ne vsebuje in so še vedno shranjeni tisti iz prvega sporočila. Pravilen pristop k prebiranju in uporabi podatkov zgleda takole:

```
void loop()
{
  while (Serial1.available() > 0) gps.encode(Serial1.read());
  if (gps.location.isUpdated())
  {
    float latitude = gps.location.lat();
    float longitude = gps.location.lng();
  }
...
} // end loop()
```

S funkcijo **isUpdated()** se pred uporabo dokončno prepričamo, ali smo prejeli nove podatke.

4.1.2 Zaznavanje delovanja stroja



Slika 4.2: Povezava senzorja za zaznavanje napetosti z mikrokrmilnikom

Zaznavanje delovanja stroja je izvedeno tako, da v senzor za zaznavanje napetosti iz stroja napeljemo žico iz stikala, kjer steče električni tok, ko obrnemo ključ za delovanje na stroju, in jo vstavimo v VCC vhod na senzorju. Za tem moramo povezati še žico za ozemljitev na stroju, ki jo vstavimo v GND vhod na senzorju. Povezava je slikovno prikazana v podpoglavju **??**, kjer vidimo praktični primer vgradnje vgrajenega sistema.

Povezave med senzorjem in mikrokrmilnikom

Na Sliki 4.2 je prikazana vezava senzorja za zaznavanje napetosti na mikrokrmilnik. Povezave so tri:

- z oranžno žico imamo povezavo na 5V električni vir,
- z rumeno žico imamo povezavo na GROUND oziroma ozemljitev in
- z rdečo žico imamo povezavo na analogni vhod na mikrokrmilniku.

Programska implementacija

Programska implementacija je zasnovana tako, da beremo vrednosti, ki prihajajo na analogni vhod 0, kar storimo s funkcijo **analogRead(0)**. Te vrednosti so razpona 0 V – 5 V, čeprav na senzor iz stroja prihajajo višje napetosti. Pri implementaciji jih moramo zato pri branju analognega vhoda prevesti v vrednosti, ki dejansko prihajajo na senzor. Ker gre pri senzorju za 5:1 napetostni delilnik, ki uporablja 30 kOhm in 7.5 kOhm upor, bomo vrednosti prevedli z nekaj enostavne matematike. Pri vsem skupaj moramo upoštevati, da nam analogni vhod vrača vrednosti 0–1023. Zato najprej prevedemo vrednost, ki jo dobimo v analogni vhod, v dejansko napetost in nato še izračunamo napetost, ki je prisotna pred upori na senzorju:

```
analogValue = analogRead(0);
voltage_on_analog = (analogValue * 5.0) / 1024.0;
voltage_on_sensor = voltage_on_analog / (7500.0/(30000.0+7500.0));
```

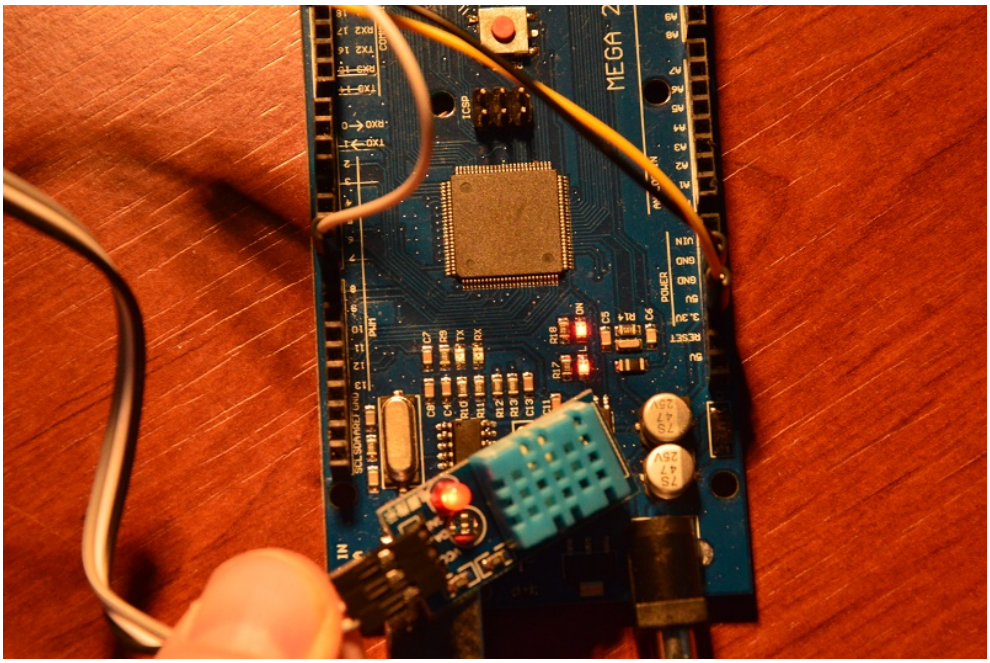
Preostane nam samo še to, da preverimo, ali je napetost prisotna in ali ta presega 11,5 V. Če presega, pomeni, da je stroj v delovanju, če pa je izmerjena napetost manjša, pa pomeni, da je z baterijo verjetno nekaj narobe in je potrebno to sporočiti.

4.1.3 Merjenje temperature in vlažnosti

Povezave med modulom in mikrokrmilnikom

Na Sliki 4.3 vidimo povezave med mikrokrmilnikom in senzorjem za temperaturo in vlažnost:

- z rumeno žico na 5 V električni vir,



Slika 4.3: Povezava senzorja za merjenje temperature in vlažnosti z mikrokrmilnikom

- s črno žico GROUND oziroma ozemljitev in
- z belo žico na digitalni vhod na mikrokrmilniku.

Programska implementacija

Pri programski implementaciji smo si pomagali s knjižnico „dht11.h“. Za branje vrednosti uporabimo funkcijo **read()**, ki vrača tri vrednosti:

- 0 – z dobljenimi podatki je vse v redu,
- -1 – napaka pri podatkih, ki so bili dobljeni, a so morda napačni in
- -2 – prišlo je do napake v komunikaciji (timeout) med mikrokrmilnikom in modulom.

Branje podatkov iz modula:

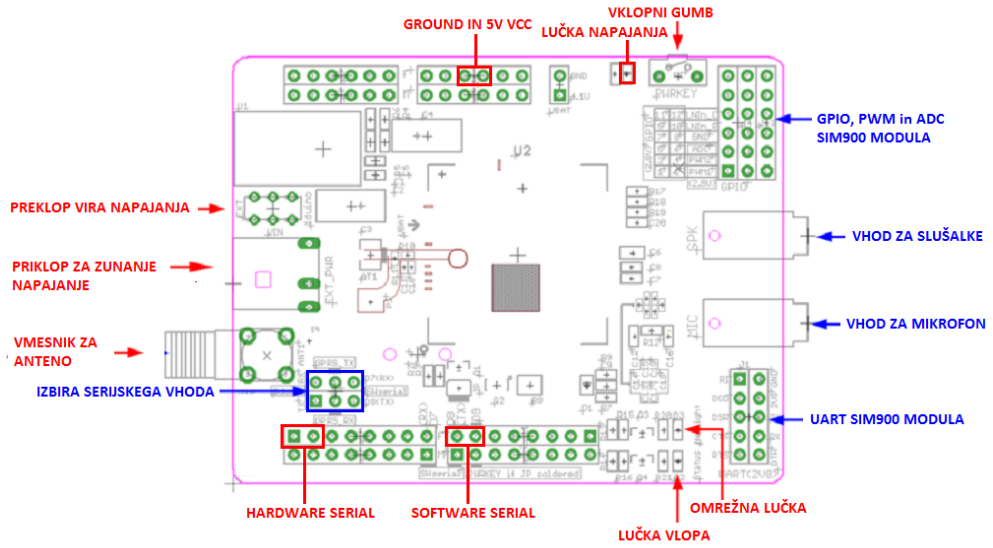
```
dht11 DHT11;

void loop()
{
...
String dht11Error;
switch(DHT11.read(7))
{
    case 0:
temperature = (float)DHT11.temperature;
humidity = (float)DHT11.humidity;
break;
    case -1:
dht11Error="Checksum error";
break;
    case -2:
dht11Error="Time out error";
break;
    default:
break;
  }
...
}
```

4.1.4 Uporaba GPRS modula

Povezave med modulom in krmilnikom

Povezati moramo dva oziroma štiri (odvisno od izbire načina napajanja) pine na GPRS shieldu in sicer:

- pin za serijski vhod,
- pin za serijski izhod,



Slika 4.4: GPRS shield Arduino UNO shema

- VCC pin za napajanje modula, ki je lahko 5 V preko 5 V pina ali 6,5 V – 12 V preko VIN pina in
- *ground* pin za ozemljitev.

Pine za serijski vhod in serijski izhod lahko izberemo na dva načina: prvi je, da pina povežemo na „HARDWARE SERIAL“, drugi pa na pina 11 in 10, prek katerih s pomočjo knjižnice softwareserial.h realiziramo „SOFTWARE SERIAL“ povezavo. Glede na izbiro vezave moramo ustrezno nastaviti pine, ki predstavljajo izbiro serijskega vhoda (glej shemo na Sliki 4.4). Izbiro izvedemo na način, da postavimo priloženi zatič tako, da na sprednja dva pina povežemo sosednja, in sicer pina na levi v kolikor želimo uporabljati „SOFTWARE SERIAL“, in pina na desni, če se odločimo za „HARDWARE SERIAL“.

Za napajanje imamo prav tako dva možna načina. Prvega predstavlja zunanje napajanje, pri katerem na priklop za zunanje napajanje (glej shemo na Sliki 4.4) priklopimo kabel. Druga možnost je, da iz mikrokrmilnika povežemo kabla za VCC 5 V napajanje in ozemljitev ter ju priklopimo na

pina označena z „GROUND IN 5V VCC“ na shemi na Sliki 4.4. Glede na izbiro načina napajanja moramo premakniti stikalo, ki je na shemi označeno z napisom „PREKLOP VIRA NAPAJANJA“. Če je z napajanjem vse v redu, bo zasvetila „LUČKA NAPAJANJA“. SIM kartico vstavimo na spodnjo stran komponente. Ko imamo vezave pravilno narejene, lahko komponento vključimo s pritiskom vklopnega gumba (glej shemo na Sliki 4.4). Po pritisku morata, če je vse v redu, zasvetiti lučka vklopa in lučka omrežja ki začne utripati enkrat na sekundo, kar pomeni, da omrežje ni najdeno. Ko modul SIM900 zazna in se prijavi v omrežje začne lučka utripati trikrat na sekundo.

Programska implementacija

Z modulom moramo najprej vzpostaviti serijsko komunikacijo. S pomočjo knjižnice SoftwareSerial.h odpremo serijsko komunikacijo na pinih 11 in 10.

```
SoftwareSerial GPRS(11,10);
```

S serijsko komunikacijo prek pinov 11 in 10 pošiljamo in prejemamo sporočila iz modula, ki ga tako krmilimo s pošiljanjem **AT ukazov**:

```
GPRS.println("‘AT"’);
confirmCommand(3000);


int confirmCommand(unsigned long timeOut)
{
  unsigned long tOut=millis();
  while((millis() - tOut) <= timeOut)
  {
    if(GPRS.available())
    {
      Serial.write(GPRS.read());

    }
  }
```

```
}
```

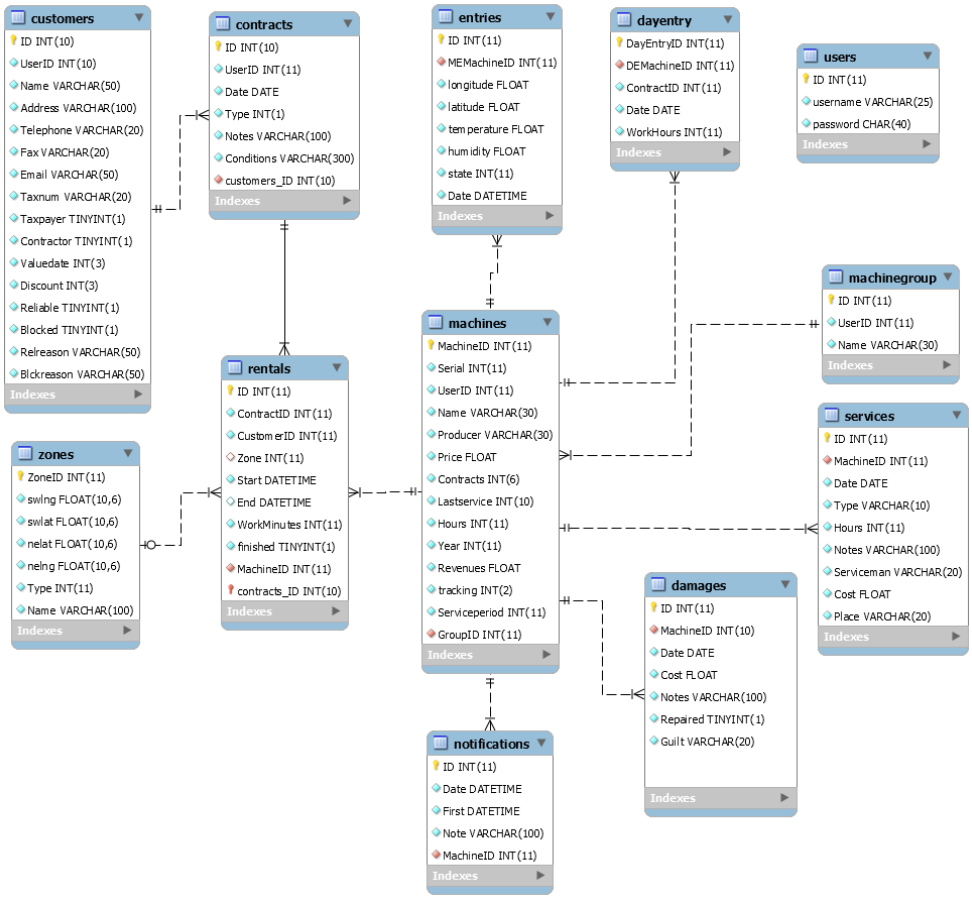
Potrjevanje ukazov je potrebno, saj moramo počakati na odgovor iz modula, preden pošljemo nov ukaz, v nasprotnem primeru lahko pride do prekrivanja ukazov in nepravilnega delovanja. Nabor AT ukazov za SIMComm-ov SIM900 modul je na voljo na:
`http://http://simcom.ee/documents/?dir=SIM900`
Uporaba AT ukazov v implementiranem vgrajenem sistemu pri povezovanju v internet in pošiljanju podatkov je predstavljena podrobneje v podpoglavju 4.3.3, kjer je podrobneje opisana komunikacija med gradbenim strojem in oblakom.

4.2 Oblačna aplikacija z nadzorno ploščo

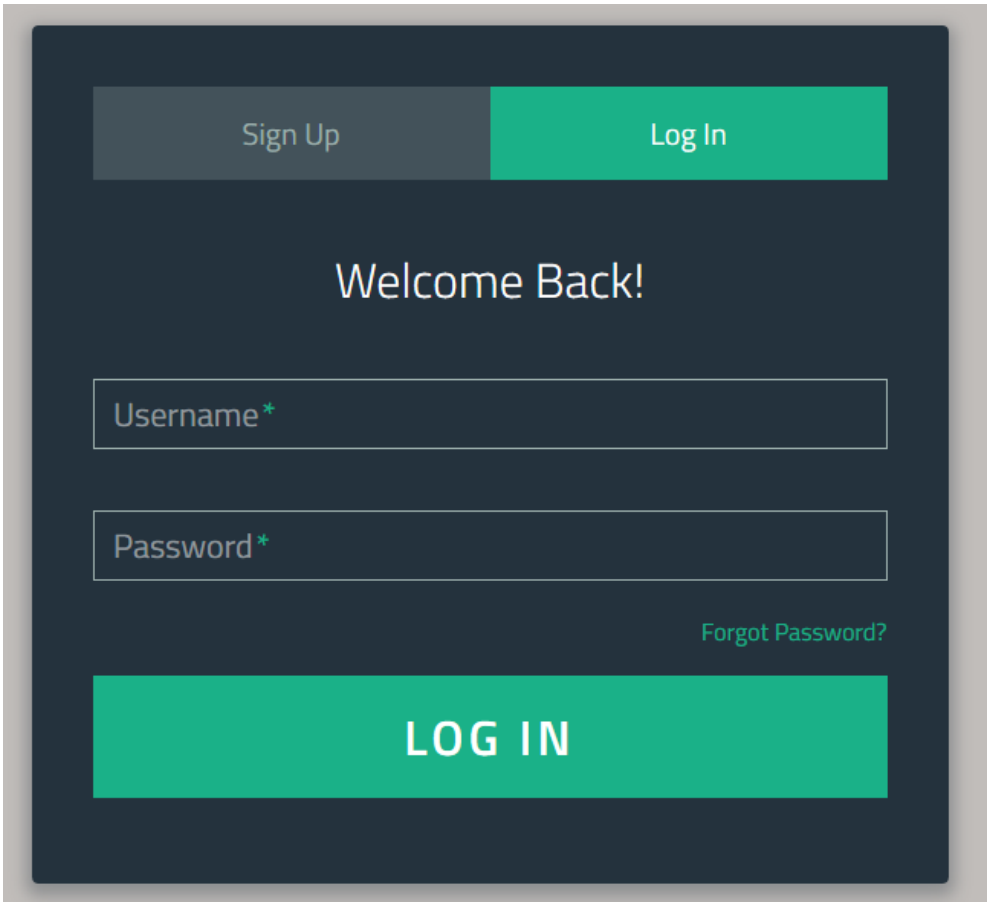
V poglavju je predstavljena implementacija nadzorne plošče sistema v oblaku s tehnologijami, ki smo jih omenili v podpoglavju 3.3.

4.2.1 Podatkovna baza

Na začetku razvoja aplikacije smo najprej pripravili shemo podatkovne baze. Glede na pričakovanja od aplikacije, opisana v 3.1, vidimo, da ima baza v našem primeru osrednjo vlogo saj v njej hranimo vse podatke pridobljene iz vgrajenih sistemov. Te hranimo v tabelah **entries**, kjer so zbrana vsa sporočila iz strojev. Sporočila vsebujejo podatke o GPS položaju, temperaturi in vlažnosti, minute delovanja in čas prejetja sporočila. V tabeli **dayentry** hranimo podatke posameznega dne s številom delovnih minut. Shema baze s tabelami in povezavami je prikazana na Sliki 4.5.



Slika 4.5: Shema podatkovne baze oblačne aplikacije



Slika 4.6: Okno za prijavo v nadzorno ploščo

4.2.2 Prijava v sistem

Uporabnik se mora pred dostopom do nadzorne plošče najprej prijaviti v sistem. To stori prek obrazca prikazanega na Sliki 4.6, v katerega vnese svoje uporabniško ime in geslo. Obrazec nato pošlje POST zahtevo na strežnik, kjer preverimo ali uporabnik obstaja. Če obstaja, preverimo še njegovo geslo, ki ga v podatkovni bazi zaradi varnosti hranimo z razpršitveno funkcijo SHA1. Če se geslo ujema, se uporabniku dodeli sejno spremenljivko veljavnega uporabnika. Po uspešni prijavi uporabnika preusmerimo na nadzorno ploščo, ki se nahaja na podnaslovu /**gui**:

4.2.3 Nadzorna plošča

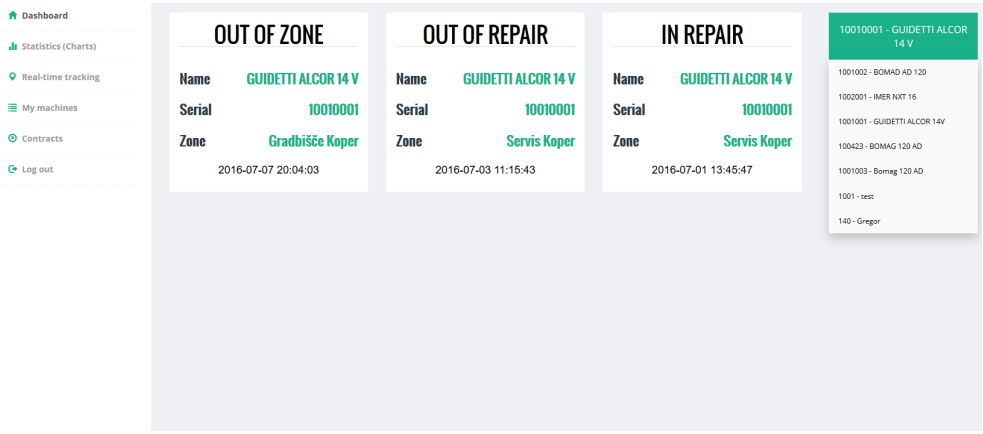
Nadzorna plošča je v našem primeru celoten uporabniški vmesnik, ki uporabniku omogoča lažji pregled in manipulacijo s podatki, ki jih hranimo v podatkovni bazi. Vmesnik smo razdelili na pet podstrani:

- domačo stran, kjer se nam izpišejo obvestila od najnovejšega do najstarejšega,
- stran z grafičnim prikazom uporabe strojev,
- realno časovno sledenje strojev,
- pregled nad zgodovino servisov, najemov, okvar in prihodkov strojev in
- pregled nad pogodbami in strankami.

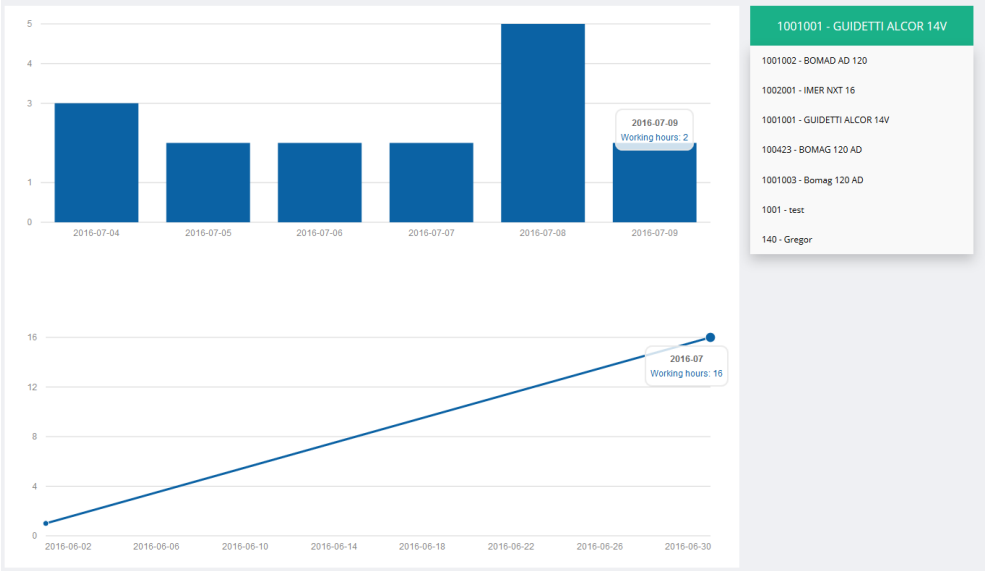
Domača stran z obvestili

Na prvi podstrani imamo pregled nad obvestili (Slika 4.7), ki jih v podatkovni bazi hranimo v tabeli **notifications**. Obvestila so prikazana v kronološkem vrstnem redu od najnovejšega do najstarejšega. V desnem meniju lahko izberemo pregled za posamezni stroj. V obvestilih hranimo naslednje dogodke:

- napake GPS signala,
- izhod in prihod stroja iz dodeljenega območja oziroma v dodeljeno območje,
- obvestilo o prekoračitvi predvidenega konca izposoje,
- začetek in konec popravil ter
- obvestilo o izgubi signala oziroma o tem, da stroj ni oddal signala ob predvidenem času glede na nastavljeno periodo pošiljanja podatkov.



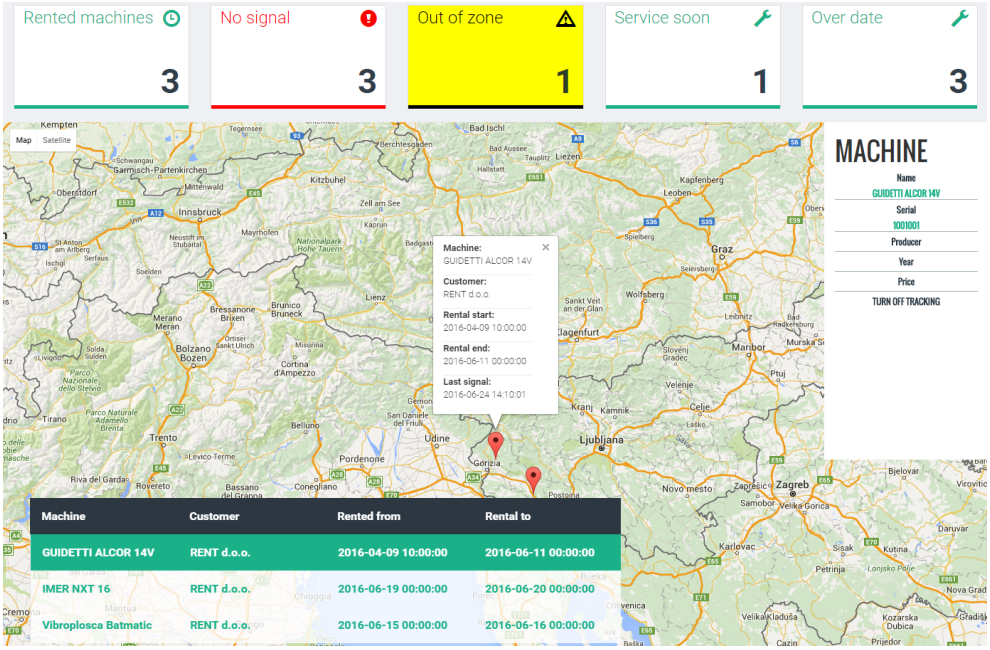
Slika 4.7: Domača stran s pregledom obvestil



Slika 4.8: Grafični prikaz uporabe strojev.

Grafični prikaz uporabe strojev

Do grafičnega pregleda uporabe strojev dostopamo prek podstrani **/stats**, kjer najprej na desni strani izberemo želeni stroj. Po izbiri se nam izrišeta dva grafa, kjer nam prvi kaže uporabo zadnjih sedem dni, drugi pa uporabo po mesecih (Slika 4.8). Grafični prikaz je bil implementiran z uporabo knjižnice **Morris.js**, ki vsebuje več objektov za izris grafičnih prikazov statistike. Pri prvem smo uporabili objekt **Bar**, pri drugem pa **Line**. Podatke izrisujemo s funkcijo **loadBarData()** z AJAX klicem na skripto **bar_data.php**, ki naredi poizvedbo v bazi in nam vrne JSON dokument s podatki, kjer imamo ključa 'date' in 'workhours'.



Slika 4.9: Sledenje strojev preko Google Maps API-ja

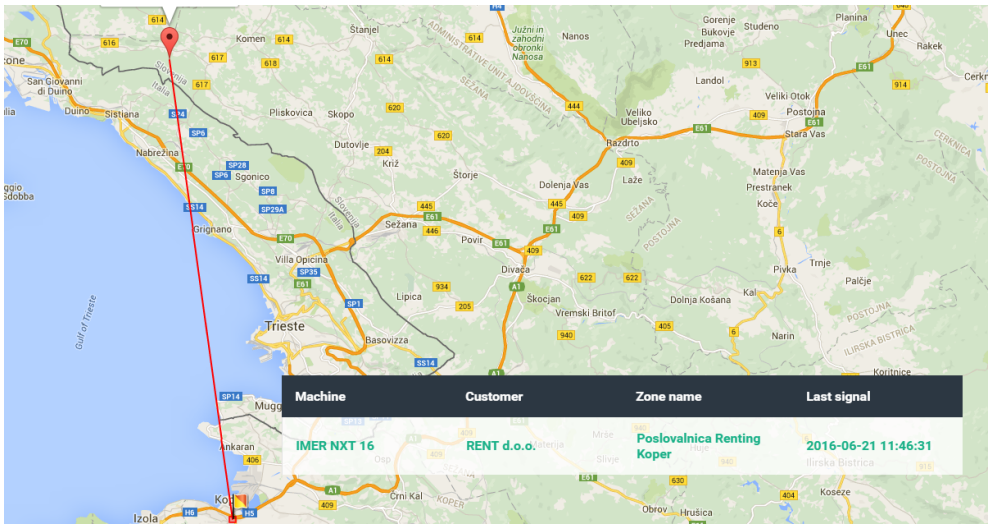
Realno časovno sledenje

Realno časovno GPS sledenje gradbenim strojem je osrednja funkcionalnost oblačne aplikacije, do katere dostopamo na podnaslovu **/track**. Izgled podstrani je prikazan na Sliki 4.9.

Položaj strojev je podan z aplikacijskim programskim vmesnikom **Google Maps Javascript API v3**, ki je za nekomercialno uporabo brezplačen, za komercialno pa ga moramo registrirati s t. i. „premium plan“-om. Za uporabo API-ja smo morali na strani https://developers.google.com/maps/ najprej pridobiti API KEY, s katerim smo lahko začeli vmesnik uporabljati. API KEY vključimo v projekt sledeče:

```
<script src="https://maps.googleapis.com/maps/api/
js?key\textbf{NAŠ API KEY} "></script>
```

Ustvarili smo 5 različnih pogledov za sledenje glede na stanje, v katerem se stroj nahaja. Pogled izberemo s pritiskom na pravokotnik nad mapo



Slika 4.10: Pregled nad stroji, ki se nahajajo izven dodeljenega območja

(Slika 4.9), v katerem je pripisano tudi število strojev v posameznem pogledu. Možnosti pogledov so:

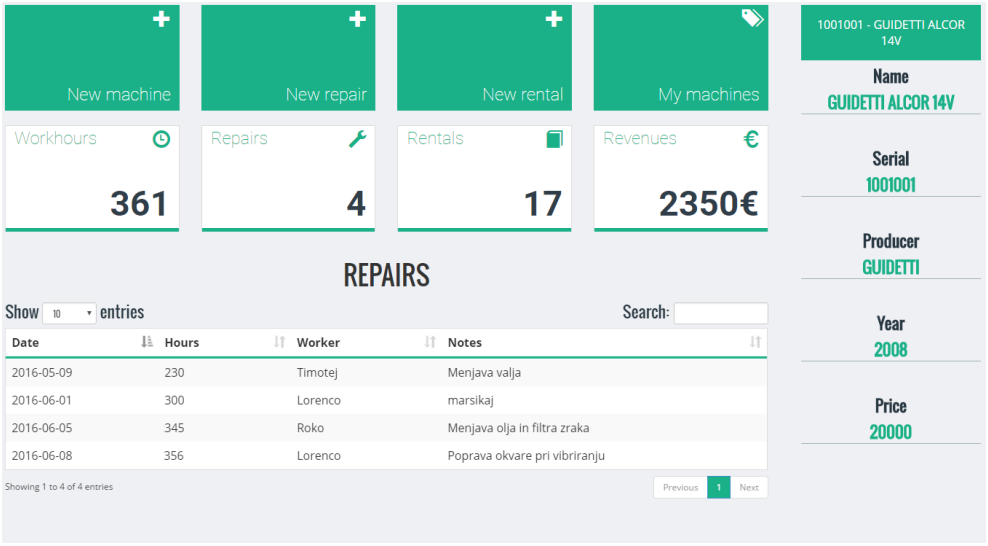
- pregled nad vsemi izposojenimi stroji z vključenim sledenjem,
- pregled nad stroji, ki dalj časa niso oddali signala,
- pregled nad stroji, ki niso v svojem dodeljenem območju,
- pregled nad stroji, ki bodo kmalu imeli redni servis glede na ure delovanja in
- pregled nad stroji, ki so prekoračili predviden datum konca izposoje.

Po izbranem pogledu se na mapi izrišejo znaki (markerji), ki nam prikazujejo položaj stroja. Za prikaz se najprej izvede AJAX klic, ki nam iz baze podatkov vrne podatke o strojih, ki ustrezajo pogledu. Primer poizvedbe za prikaz strojev, ki se nahajajo izven dodeljenega območja:

```
SELECT * FROM rentals AS r
INNER JOIN machines m ON r.MachineID=m.MachineID
```

```
INNER JOIN customers AS c ON r.CustomerID=c.ID
INNER JOIN vnosi AS v ON v.MEMachineID=m.MachineID
LEFT JOIN zones AS z ON z.ZoneID=r.Zone
WHERE r.ID IN (SELECT MAX(ID) FROM rentals GROUP BY MachineID)
AND v.ID IN (SELECT MAX(ID) FROM vnosi GROUP BY MEMachineID)
AND (r.Zone IS NOT NULL)
AND (v.longitude NOT BETWEEN z.swlng AND z.nelng)
OR (v.latitude NOT BETWEEN z.swlat AND z.nelat)
AND r.finished=0
ORDER BY v.Date DESC
```

Za izris na mapi je implementirana Javascript funkcija **showMachine()**, ki za vsak stroj izriše marker in ustvari objekt **infoWindow**, ki nam ob kliku na marker (funckija addListener) ustvari fokus na stroj in poda dodatne informacije o njem, stranki, ki ga ima v izposoji, trajanju izposoje in času zadnjega prejetega signala. Posamezen stroj lahko izberemo tudi v dodatni tabeli, ki nam izpiše, kateri stroji vse so prikazani v izbranem pogledu.
Za pregled nad stroji ki se ne nahajajo v dodeljenem območju (Slika 4.10), smo implementirali tudi funkcijo **showZone()**, ki nam v obliki rdečega pravokotnika z zastavico prikaže dodeljeno območje, in funkcijo **zoneToMachine()**, ki izriše črto med strojem in dodeljenim območjem.



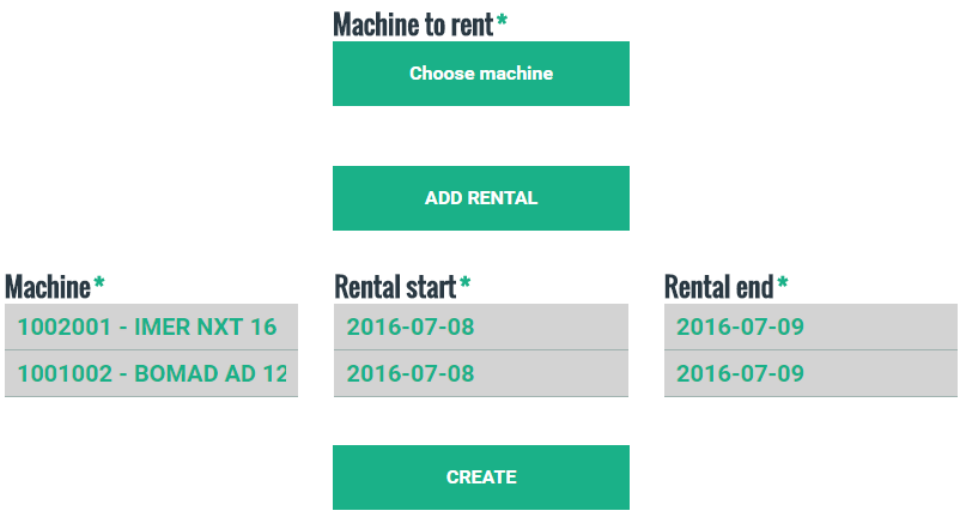
Slika 4.11: Pregled nad popravili strojev

Pregled nad stroji

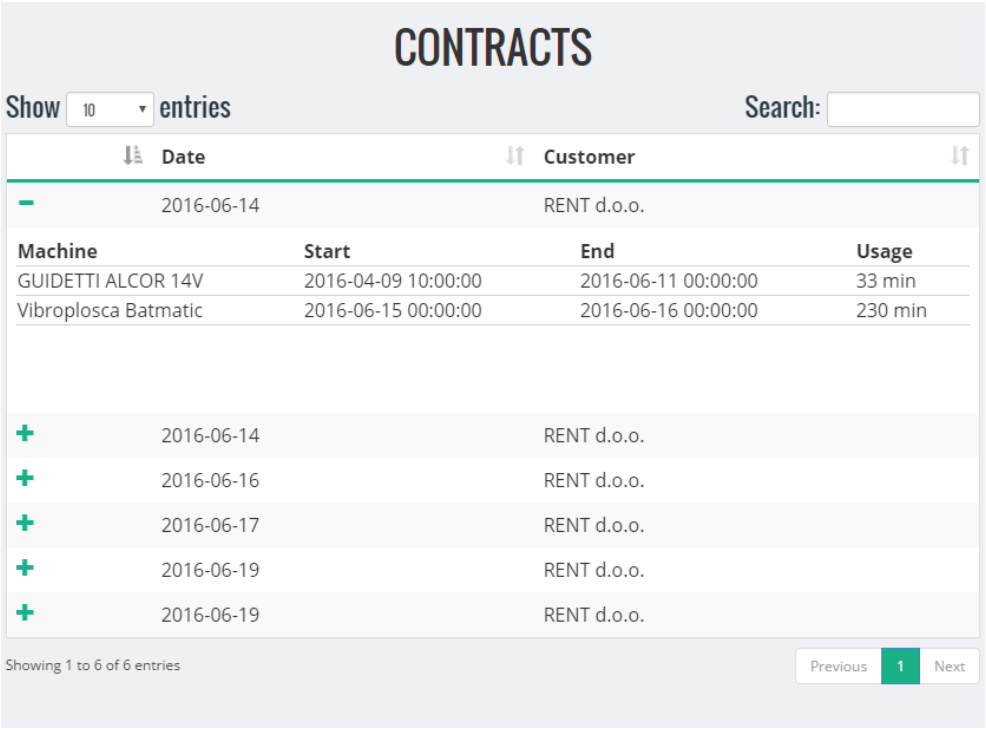
Ker želimo imeti tudi popoln splošen pregled nad stroji, ki jih izposojamo, imamo na podstrani /machines tabelarični pregled nad popravili, delovnimi urami, izposojami in prihodki za vsak posamezen stroj. Pregled nad popravili na stroju je prikazan na Sliki 4.11. Na isti podstrani tudi dodajamo nove stroje.

Pogodbe in stranke

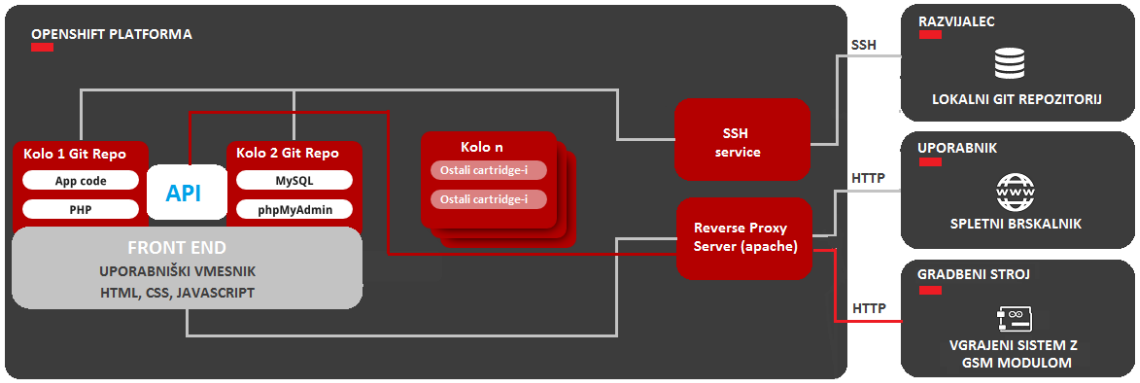
Za uspešno poslovanje z izposojanjem strojev moramo imeti pregled tudi nad strankami, ki si naše stroje izposojajo, in pogodbami. Na Sliki 4.12 je prikazan obrazec za dodajanje novih pogodb. Najprej vnesemo podatke o pogodbi, nato pa dodamo željene izposoje. Tako dobimo koncept, kjer mora biti za izposojo stroja obvezno narejena pogodba, ki lahko vsebuje več izposoj. Na podoben način je narejeno tudi pregled nad strankami in dodajanje novih. Pregled nad izdanimi pogodbami in izposojami, ki so bile izvedene v okviru posamezne pogodbe, je prikazan na Sliki 4.13.



Slika 4.12: Dodajanje nove izposoje.



Slika 4.13: Pregled nad pogodbami



Slika 4.14: Shema komunikacij na platformi

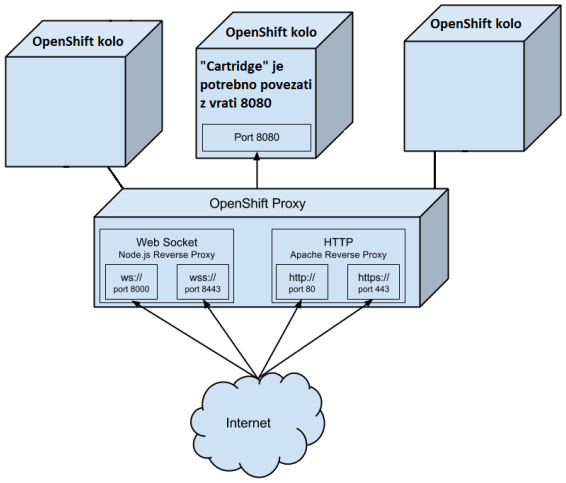
4.3 Komunikacija v sistemu

Komunikacijsko shemo sistema vidimo na Sliki 4.14. Obravnavamo lahko tri različne tipe komunikacije z oblakom, in sicer:

- komuniciranje razvijalca aplikacije z oblakom,
- komuniciranje končnega uporabnika z oblakom in
- komunikacija med gradbenim strojem in oblakom.

4.3.1 Komuniciranje razvijalca aplikacije s sistemom

Kot smo omenili že v podpoglavju 3.3.1, nam platforma OpenShift omogoča, da naredimo kopijo repozitorija v oblaku, kjer imamo datoteke aplikacije, in razvijamo dalje v lokalnem repozitoriju. Preko Git-a enostavno prenesemo v oblak. Kot vidimo na sliki arhitekture sistema, je to izvedeno preko SSH protokola, kar nam omogoča OpenShift-ov **SSH service**.



[ht!]

Slika 4.15: Openshift reverse proxy server

4.3.2 Komuniciranje končnega uporabnika s sistemom

Uporabniki sistema dostopajo do uporabniškega vmesnika s spletnim brskalnikom in sicer prek HTTP oziroma HTTPS (vrata 80 in 443) ali Web Socket oziroma Web Socket Secure protokola (vrata 8000 in 8443). Uporabnik prek omenjenih protokolov dostopa najprej do t. i. **OpenShift Proxy**-ja (Slika 4.15). Za HTTP in HTTPS dostopa je uporabljen Apache reverse proxy server (navaden Apache strežnik z modulom mod_proxy), za Web Socket pa Node.js reverse proxy server. Glavna naloga reverse proxy strežnikov je usmerjanje zahtev med kolesi. Predvsem v oblaku ti strežniki opravljajo tudi nalogo „Load balancer“-jev (uravnoteženja bremen), kar z mod_proxy-em dosežemo z metodo **mod_proxy_balancer**, ki služi uravnoteženju bremen prek več „back-end“ entitet. Mod_proxy nam, kot smo že omenili, omogoča tudi SSL tuneliranje, in sicer z metodo mod_proxy_connect.
OpenShift uporablja tudi Port Proxy, ki vsebuje iptables (program, ki hrani povezave in pravila za povezovanje preko IP naslovov) in služi komunikaciji med kolesi znotraj aplikacije ter je tako ključni del omogočanja skalabilnosti.

4.3.3 Komunikacija med gradbenim strojem in oblakom

Komunikacija med gradbenim strojem oziroma vgrajenim sistemom v njem in oblakom prav tako poteka prek HTTP protokola. V podpoglavju 2.4 smo v okviru interneta stvari omenili več možnosti pri izbiri protokola. Izbran je bil HTTP protokol, pri čemer je bilo prvo merilo TCP protokol na transportni plasti, saj želimo, da je prenos podatkov, ki jih pošiljamo iz strojev, zagotovljen. Tako je protokol CoAP odpadel. Drugo merilo je bilo, da je protokol tipa request/response, saj publish/subscribe protokola MQTT in XMPP vzdržujeta odprto TCP povezavo. V našem primeru, ko podatke pošiljamo periodično na vsako minuto oziroma na vsakih 15 ali 30 minut, je tak pristop nesmiseln. Slednja prideta v poštev predvsem za t. i. realnočasovno komunikacijo, kjer so na senzorjih nenadne spremembe, ki jih želimo takoj zaznati, saj omenjena protokola zagotavljata precej manjšo latenco kot HTTP protokol. V spodnjih dveh podpoglavjih je prikaz realizacije komunikacijskega vmesnika na vgrajenem sistemu in API-ja oblaku.

```
{
   "mid": 9,
   "temp": 15,
   "longt": 13.689982,
   "latd": 45.804729,
   "hum": 55,
   "state": 1
}
```

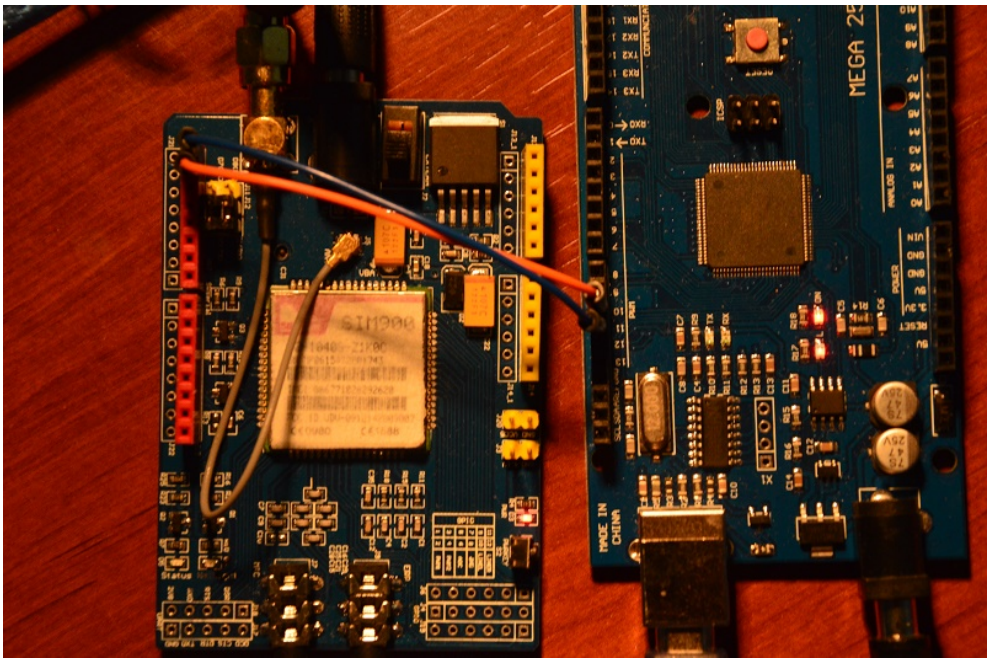
```
String PostData="{ \"mid\": ";
PostData.concat(id);
PostData.concat(", \"temp\": ");
PostData.concat(temperature);
PostData.concat(", \"longt\": ");
PostData.concat(lngBuff);
PostData.concat(", \"latd\": ");
PostData.concat(latBuff);
PostData.concat(", \"hum\": ");
PostData.concat(humidity);
PostData.concat(", \"state\": ");
PostData.concat(state);
PostData.concat(" }");
```

Slika 4.16: Primer pristnega JSON-a in njegova tvorba v mikrokrmilniku

Podatke iz gradbenih strojev pošiljamo v oblak tako, da vgrajeni sistem v oblak pošlje POST zahtevo s pristnim JSON dokumentom. Ta vsebuje podatke o GPS koordinatah, temperaturi, vlažnosti, ID stroja in polje stanja, ki nam pove, koliko minut je stroj deloval. Primer pristnega JSON dokumenta

je prikazan na Sliki 4.16. Bistvo JSON-a poslanega iz vgrajenega sistema je njegova enostavnost in vsebovanost bistvenih podatkov, saj želimo, da je sporočilo kar se da kratko, ker tako ne porabljamo nepotrebne energije.

Pošiljanje podatkov iz vgrajenega sistema



Slika 4.17: Povezava GPRS shield-a z mikrokrmilnikom

Iz gradbenega stroja podatke pošiljamo prek GPRS shielda, ki smo ga podrobneje opisali v podpoglavju 3.2. Za krmiljenje modula smo morali najprej vzpostaviti komunikacijo med GPRS shield-om in mikrokrmilnikom.
Po vzpostavljeni povezavi z modulom kličemo funkcijo **establishConnection()**, s katero se prek pošiljanja AT ukazov modulu povežemo v GPRS omrežje.

```
void establishConnection()
{
  //APN Telekoma Slovenije
```

```
String MY_APN="internet";

  //Nastavitev tipa omrežja
GPRS.println("AT+SAPBR=3,1,\"Contype\",\"GPRS\"");
  confirmCommand(3000);
//Nastavitev APN
  GPRS.print("AT+SAPBR=3,1,\"APN\",\"");
  GPRS.print(MY_APN);
  GPRS.println("\"");
  confirmCommand(3000);

//Preverimo ali je s povezavo vse v redu
  GPRS.println("AT+SAPBR=2,1");
  confirmCommand(3000);
}
```

Ko imamo povezavo vzpostavljeno, lahko začnemo s pošiljanjem zahteve strežniku. Najprej pridobljene podatke iz senzorjev sestavimo v JSON dokument, ki ga pošljemo na strežnik v obliki POST zahteve s funkcijo **sendRequest()**:

```
void establishConnection()
{
  String MY_URL="http://track-stopar.rhcloud.com/api/api.php";


GPRS.println("AT+HTTPINIT");
  confirmCommand(3000);

GPRS.println("AT+HTTPPARA=\"CID\",1");
  confirmCommand(3000);
  GPRS.print("AT+HTTPPARA=\"URL\",\"");
  GPRS.print(MY_URL);
```

```
  GPRS.println("\"");
  confirmCommand(3000);
  GPRS.print("AT+HTTPPARA=\"CONTENT\",\"");
  GPRS.print("application/json");
  GPRS.println("\"");
  confirmCommand(3000);
  GPRS.print("AT+HTTPDATA=");
  GPRS.print(PostData.length());
  GPRS.println(",10000");
  confirmCommand(3000);
  GPRS.println(PostData);
  confirmCommand(3000);
  GPRS.println("AT+HTTPACTION=1");
  confirmCommand(3000);
  GPRS.println("AT+HTTPREAD");
}
```

Po ukazu **AT+HTTPREAD** čakamo na odgovor strežnika. Implementirali smo funkcijo za branje odgovorov **readResponse()**, ki razčleni odgovor modula na AT+HTTPREAD ukaz, znotraj katerega je tudi odgovor iz strežnika. Odgovor nam pove, ali je bilo s podatki vse v redu in s kakšno periodo naj v prihodnje pošiljamo podatke. Primeri sporočil: **ok&1**, **ok&15**, **gpserror&1** ,...
Sporočila v mikrokrmilniku preberemo tako:

```
GPRS.println("AT+HTTPREAD");
  sendperiod=readResponse(5000);
```

Prejemanje zahtev v oblaku

Na strežniku zahteve prejemamo na aplikacijskem programskem vmesniku (API – aplication program interface) na spletnem naslovu:

$$http://track-stopar.rhcloud.com/api/api.php \tag{4.1}$$

Skripta obravnava samo POST zahteve, kjer najprej preverimo pristnost prejetih podatkov. V kolikor je s podatki nekaj narobe, na primer, če nismo dobili GPS koordinat ali so koordinate stroja izven območja, ki smo ga za stroj določili, v bazo shranimo obvestilo z opisom. Če je s podatki vse v redu jih vnesemo v bazo podatkov s SQL transakcijo:

```
BEGIN;
  INSERT INTO dayentry (DEMachineID, Date, WorkHours)
VALUES (:mid, CURDATE(), :state)
ON DUPLICATE KEY UPDATE WorkHours=WorkHours+:state

  INSERT INTO entries
  VALUES (0, :MachineID, :longitude, :latitude,
:temperature, :humidity, :state, now())
COMMIT;
```

Tako shranjujemo podatke v tabelo dnevnih vnosov **dayentry**, kjer imamo unikatni ključ za eno dnevno vrednost na stroj. Ob drugem vnosu istega dne samo povečujemo stanje delovnih ur. V drugo tabelo **entries** shranjujemo vse vnose, ki jih prejmemo iz gradbenega stroja.
Podatke v tabeli entries se po koncu izposoje pobrišejo.

4.4 Praktični primer

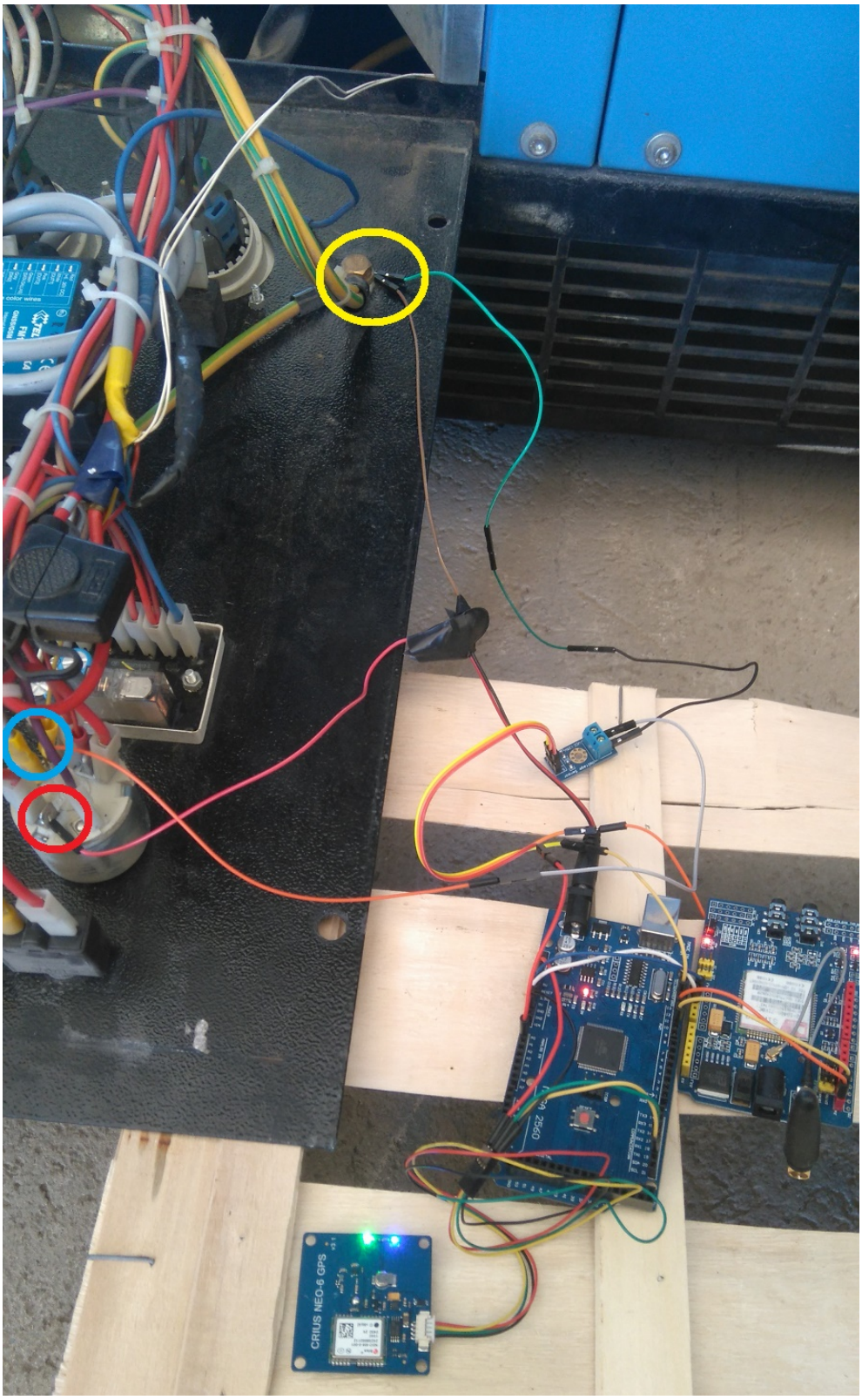
Slika 4.18: Električni generator GENSET MG 33

Za prikaz praktičnega primera smo uporabili električni generator GENSET MG 33 na Sliki 4.18.

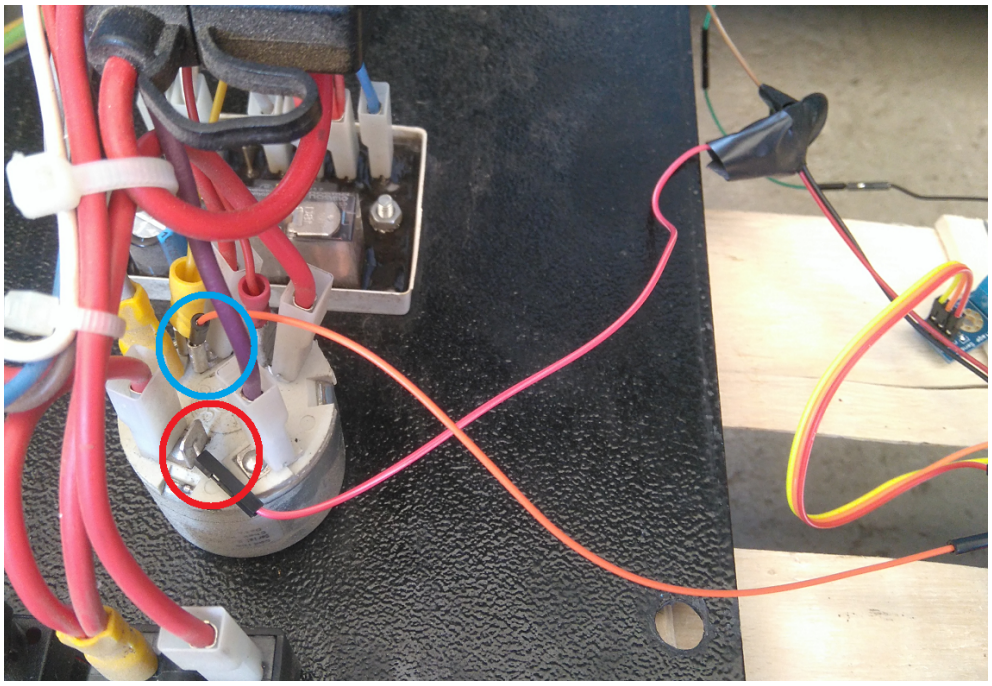
4.4.1 Vgradnja vgrajenega sistema

Slika 4.19: Odstranitev nadzorne plošče za vgradnjo sistema

Mikrokrmilnik s senzorji in z moduli smo vgradili tako, da smo najprej odprli nadzorno ploščo (Slika 4.19). Za vgradnjo smo morali iz stroja dobiti vir napajanja za vgrajeni sistem in povezati senzor za zaznavanje napetosti s stikalom, ki se aktivira, ko obrnemo zagonski ključ na stroju. Tako žico za napajanje kot tudi žico, ki jo bomo vstavili v senzor bomo priklopili na zagonsko stikalo. Zadnji del stikala je prikazan na Sliki 4.21. Del, ki je označen z rdečim krogom, ima stalni vir napetosti in je uporabljen za napajanje. Nanj smo priklopili rdečo žico, ki je vezana na priključek za napajanje. Del, ki je označen z modrim dobi vir napetosti, ko obrnemo zagonski ključ in je povezana žica, ki je vstavljena v senzor za merjenje napetosti. Celoten priključen vgrajeni sistem je prikazan na Sliki 4.20. Za pravilno delovanje napajanja in merjenja napetosti moramo povezati še žici za ozemljitev, kar vidimo na sliki celega sistema označeno z rumeno barvo.

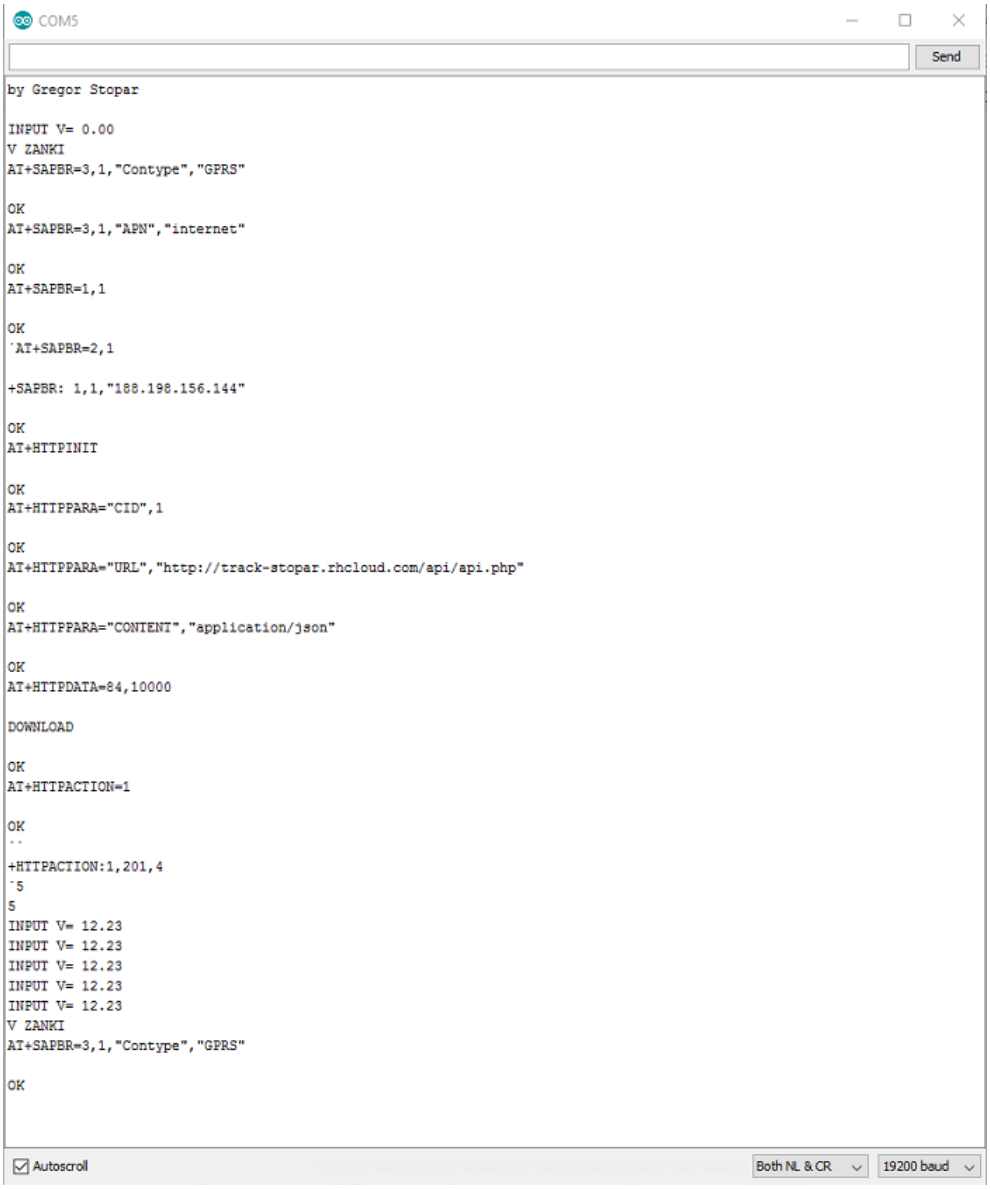


Slika 4.20: Celoten prikaz priklopljenega vgrajenega sistema



Slika 4.21: Stikalo kamor se priklopita žica za napajanje in žica za preverjanje delovanja

4.4.2 Prikaz delovanja vgrajenega sistema

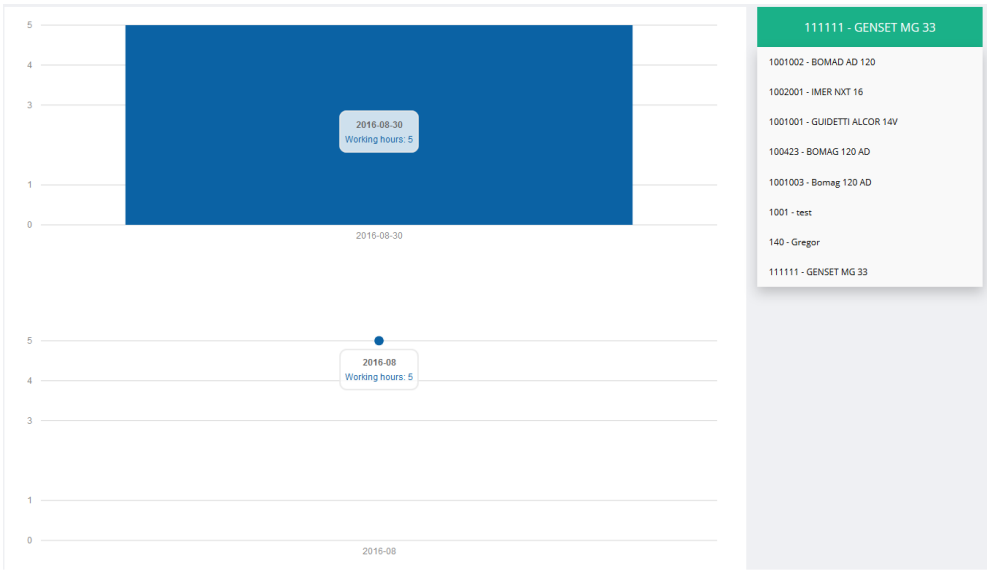


Slika 4.22: Serial monitor, ki prikazuje delovanje mikrokrmilnika

Za prikaz delovanja vgrajenega sistema smo v kodo mikrokrmilnika dodali nekaj izpisov, ki jih lahko vidimo v Arduino IDE prek t. i. „Serial monitor“-ja. S primera je razvidno, da se prvi signal pošlje takoj, tako da se

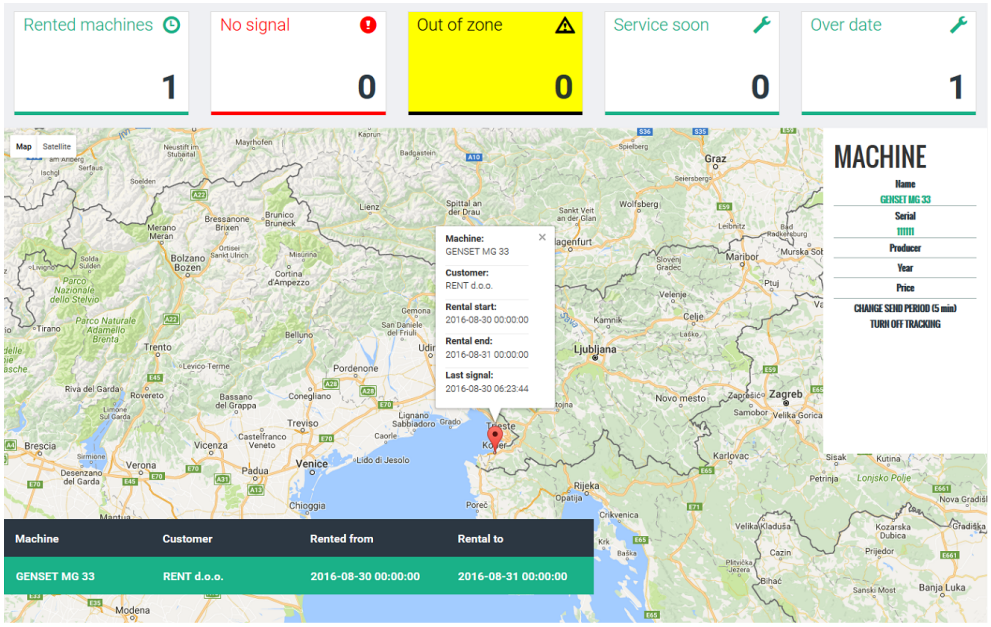
najprej vzpostavi GPRS povezava nato pa sledi pošiljanje HTTP zahtevka na strežnik. Na koncu se prebere odgovor iz strežnika z vsebovano periodo nadaljnjega pošiljanja. Nastavljena perioda pošiljanja podatkov je vsakih pet minut, medtem ko se delovanje stroja preverja vsako minuto. Tako vidimo, da po prvem poslanem signalu najprej dobimo pet izpisov stanja delovanja, kjer je ob vsakem obhodu izpisana izmerjena napetost. Po petih minutah se podatki znova pošljejo. Izpisi na Serial monitor-ju so prikazani na Sliki 4.22.

4.4.3 Prikaz delovanja nadzorne plošče



Slika 4.23: Prikaz statistike uporabe po prvem prejetem signalu

Za prikaz delovanja nadzorne plošče smo najprej naredili vnos novega stroja, ki smo ga poimenovali GENSET MG 33. Na Sliki 4.23 je prikazana statistika uporabe po prejetju prvega sporočila iz stroja. Kot smo omenili pri prikazu vgrajenega sistema, je nastavljena perioda pošiljanja 5 minut in tako vidimo, da smo v prvem signalu dobili podatek o 5 minutah delovanja, saj je bil do prvega pošiljanja stroj v delovanju. Na podstrani za realno-časovno sledenje (Slika 4.24) vidimo, da se stroj nahaja v Kopru v skladišču podjetja.



Slika 4.24: Prikaz realno-časovnega sledenja po prejetju signala iz stroja

4.5 Ugotovitve pri implementaciji

Pri implementaciji so bile prisotne določene težave, predvsem pri implementaciji določenih modulov vgrajenega sistema. Nekaj preglavic je povzročala implementacija GPS modula, kjer najprej nismo pravilno prebirali podatkov in se ti niso pravilno osveževali. Problem je bil v tem, da smo prebrali zgolj eno sporočilo iz modula in nato šli v zanko za pošiljanje. To je seveda narobe, saj modul pošlje več različnih tipov NMEA sporočil, med katerimi vsa ne vsebujejo vseh podatkov. Primer takega sporočila je sporočilo tipa GSA, ki vsebuje zgolj stanja satelitov. Napaka je privedla do tega, da so bili v sporočilih stari podatki, prebrani tri ali štiri obhode zank pred tem. Problem smo rešili s prebiranjem sporočil iz modula v zanki in preverjanjem, ali so vrednosti osvežene.
Pri implementaciji drugih komponent večjih težav ni bilo. Odprtokodna skupnost razvijalcev Arduino vgrajenih sistemov je precej velika in tako imamo na voljo veliko knjižnic in napotkov za implementacije različnih sistemov. Za izdelavo prototipnega modela je prav zato tega Arduino odlična izbira.
Kot glavno pomanjkljivost implementacije bi izpostavil velikost vgrajenega sistema. Za resnejšo komercialno uporabo bi morali imeti izdelano ploščico po naročilu, kjer bi se nahajale vse komponente vgrajenega sistema brez odvečnih neuporabljenih vhodov in izhodov.

Napajanje vgrajenega sistema

S ploščico izdelano po meri bi prihranili tudi na porabi energije. Pri večini strojev, za katere je sistem namenjen je sicer možnost napajanja modula iz akumulatorja, ki se nahaja v stroju. Tipično gre za 12 V svinčeni akumulator s kapaciteto vsaj 40 Ah. Izmerjeni tok pri delovanju Arduina z moduli je bil med 0,20 A in 0,25 A v najslabšem primeru, do koder je narasel pri vzpostavljanju povezave in pošiljanju podatkov. Če izračunamo najslabši možen scenarij, kar je pošiljanje podatkov vsako minuto, to pomeni, da mikrokrmil-

nik nenehno komunicira z GSM modulom, dobimo:

$$t = \frac{Q}{I} = \frac{40Ah}{0,25A} = 160h \tag{4.2}$$

kjer je:
$I[A]$ – električni tok
$Q[Ah]$ – električni naboj

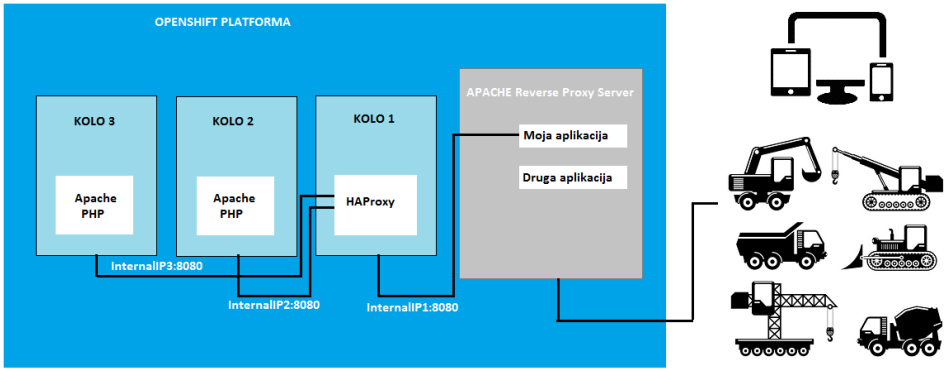
V najslabšem primeru se torej lahko zgodi, da bi delovanje vgrajenega sistema ob nedelovanju stroja izpraznilo akumulator v 160 urah oziroma nekaj več kot 6 dneh. To številko lahko bistveno povečamo z zmanjšanjem pogostosti pošiljanja podatkov iz stroja. Predvsem, ko je stroj na gradbišču v mirovanju, lahko v nadzorni plošči preklopimo na manj pogosto pošiljanje podatkov. V primeru premikanja stroja bo sistem to zaznal in samodejno preklopil na neprestano pošiljanje podatkov, obenem se bo v sistemu pojavilo obvestilo o premikanju. V vsakem primeru pa je priporočljiva pozornost, v kolikor stroj več dni ni v delovanju.

Razširljivost sistema in ocena stroškov

Za razširljivost (ang. scaling) sistema je Openshift odlična izbira. Kot smo povedali že v podpoglavju 3.3.1, OpenShift ponuja kapacitete v različnih tipih koles. Te se med seboj cenovno ločijo glede na zmogljivost:

- **Small** – 512MB RAM, 1GB disk, 0,02€/ura
- **Small.highcpu** – 512MB RAM, 1GB disk, 0,025€/ura
- **Medium** – 1GB RAM, 1GB disk, 0,04€/ura
- **Large** – 2GB RAM, 1GB disk, 0,08€/ura

Platforma je na voljo v treh različnih **„PLAN“**-ih (paketih): **FREE**, **BRONZE** in **SILVER**. Za potrebe izdelane aplikacije sem vzel brezplačen paket (FREE PLAN), ki nam nudi uporabo treh malih (Small) koles. BRONZE



Slika 4.25: Dodeljevanje koles prek HAProxy-ja

PLAN je v osnovi prav tako brezplačen, v njem dobimo tri mala kolesa zastonj, a imamo možnost razširitve na največ 16 koles, ki se plačajo po zgoraj navedenih cenah. SILVER PLAN je na voljo od 15€/mesec, za kar dobimo tri mala kolesa, kjer je za vsako kolo na voljo 6 GB diska, kapacitete pa lahko razširimo na več kot 16 koles.
Razširljivost sistema moramo omogočiti sami ob stvarjenju aplikacije, kjer za scaling izberemo „SCALE WITH WEB TRAFFIC“. Na posamezno kolo je omogočenih 16 istočasno odprtih sej. OpenShift za dodajanje potrebnih dodatnih koles poskrbi sam, kjer ob minimalnem prometu aktivira eno kolo, pri maksimalnem pa vsa razpoložljiva oziroma toliko, kolikor jih za vsak *cartridge* določimo. Openshift to stori tako, da pri aplikacijah z omogočeno razširljivostjo doda **HAProxy** *cartridge*, ki je neke vrste posrednik med aplikacijo in internetom (Slika 4.25). Njegova glavna naloga je, da opozarja Openshift o potrebnem povečanju kapacitet:

- ob vzdrževanju odprtih povezav pri 90 % zasedenoti kolesa doda novo kolo in

- v kolikor zasedenost povezav pade pod 50 % za več minut, HAProxy odstrani dodano kolo.

Kolo se doda s kopirano kodo iz Git repozitorija in s praznim podatkovnim direktorijem. Ob spreminjanju kode preko Git-a se ta obnovi v vseh kolesih. Več podrobnosti o razširljivosti sistema Openshift je dostopnih na: `https://developers.openshift.com/managing-your-applications/scaling.html`
Na ta način se tvori način uporabe „plačaj, kolikor porabiš“, ki je bil omenjen kot ena izmed osrednjih prednosti oblačnih storitev v poglavju 2.2.1. Vsak trenutek lahko svoj paket tudi nadgradimo, ne da bi pri tem morali karkoli spreminjati pri aplikaciji.
Strošek paketa za GPRS povezavo v internet preko SIM kartice Telekoma Slovenije, ki jo vstavimo v GPRS shield, je 4,50 €/mesec.
Ostali stroški so povezani z realizacijo vgrajenega sistema in znašajo:

- Mikrokrmilnik ATmega2560 6,52 €
- Crius U-blox NEO-6M V3.1 GPS modul 17,88 €
- SIM900 Quad-Band GSM/GPRS modul/shield 19,87 €
- Senzor za zaznavanje napetosti 0,89 €
- DHT11 senzor za merjenje temperature in vlažnosti 1,05 €
- Paket 60 jumper žic 4,52 €

Skupni strošek izdelanega vgrajenega sistema je 50,73 €. Z zgoraj omenjenim pomanjšanjem ploščice in masovnim naročilom bi za komercialno uporabo ta strošek lahko znižali.

Poglavje 5

Zaključek

V diplomskem delu smo naredili pregled področja, implementacijo in prikaz uporabe interneta stvari.
Pri pregledu področja smo dali poudarek na internet stvari v industriji, ki predstavlja hitro razvijajoče področje te tehnologije in za katerega velja, da prinaša velik donos glede na investicijo. Poleg tega smo naredili tudi pregled oblačnih storitev in njihovo souporabo z internetom stvari.
Pri internetu stvari smo dali poseben poudarek na komunikacijo kjer smo našteli nekatere specifične protokole, med katerimi so bili določeni narejeni posebej za uporabo v internetu stvari. Med naštetimi smo pri implementaciji lastnega sistema izbrali enega in izbiro tudi argumentirali.
Pregledali smo še pomen vgrajenih sistemov v internetu stvari, s katerimi v sistemu realiziramo senzorje za zbiranje podatkov.
Pri implementaciji lastnega sistema smo dosegli vse zadane cilje in želene funkcionalnosti v skladu s pričakovanji. Sistem je še v prototipni fazi in bi za resnejšo komercialno uporabo potreboval preoblikovanje vgrajenega sistema, ki bi moral biti predvsem manjše velikosti. Zaželena bi bila tudi zmanjšana poraba energije, saj bi tako lahko sistem napajali z manjšimi baterijami in bi bil primeren za stroje brez lastnega akumulatorja. Izdelati bi morali tudi primerno ohišje za zaščito vgrajenega sistema.
S stroškovnega vidika, lahko rečemo, da je rešitev ugodna in bi, kot smo ugo-

tovili pri ugotovitvah pri implementaciji, bila še ugodnejša ob večjih naročilih komponent. Z nekaj spremembami v aplikacijskem delu bi lahko dosegli, da rešitev ni namenjena zgolj najemodajalcem, temveč tudi izvajalnim gradbenim podjetjem, ki bi rada imela večji nadzor nad svojo mehanizacijo. Tako bi tržni potencial rešitve postal še večji.

Literatura

[1] M. Rouse, IoT Agenda, "Internet of Things (IoT)", Junij 2014. [Online]. Dosegljivo: http://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoTbtxdoc.pdf. [Dostopano 31. 3. 2016].

[2] Juniper Research, "'Internet of things' connected devices to almost triple to over 38 billion units by 2020", Julij 2015. [Online]. Dosegljivo: http://www.juniperresearch.com/press/press-releases/iot-connected-devices-to-triple-to-38-bn-by-2020. [Dostopano 31. 3. 2016].

[3] K. L. Lueth, IoT Analytics, "The 10 most popular Internet of Things applications right now", Februar 2015. [Online]. Dosegljivo: http://iot-analytics.com/10-internet-of-things-applications/. [Dostopano 31. 3. 2016].

[4] S. Lohr, The New York Times, "The Internet of Things and the Future of Farming", August 2015. [Online]. Dosegljivo: http://bits.blogs.nytimes.com/2015/08/03/the-internet-of-things-and-the-future-of-farming/. [Dostopano 31. 3. 2016].

[5] IRMS 360, "The State of the Cloud 2015: Supply Chain Adopters Reaping ROI Rewards", May 2015. [Online]. Dosegljivo: http://www.irms360.com/blog_post/state_cloud_2015_supply_chain_adopters_reaping_roi_rewards. [Dostopano 31. 3. 2016].

[6] CloudTweaks, "Cloud Deployment Models", July 2012. [Online]. Dosegljivo: http://cloudtweaks.com/2012/07/4-primary-cloud-deployment-models/. [Dostopano 31. 3. 2016].

[7] M. Rouse, Tech target, "Eucalyptus". [Online]. Dosegljivo: http://searchcloudprovider.techtarget.com/definition/Eucalyptus. [Dostopano 31. 3. 2016].

[8] Gartner IT Glossary, "Community cloud", July 2012. [Online]. Dosegljivo: http://www.gartner.com/it-glossary/community-cloud. [Dostopano 31. 3. 2016].

[9] Datacenter Knowledge, "Explaining community cloud", July 2012. [Online]. Dosegljivo: http://www.datacenterknowledge.com/archives/2014/10/13/explaining-community-cloud/. [Dostopano 31. 3. 2016].

[10] V. Beal, Webopedia, "Embedded system", July 2012. [Online]. Dosegljivo: http://www.webopedia.com/TERM/E/embedded_system.html. [Dostopano 31. 3. 2016].

[11] ALAM, Kazi Masudul; SOPENA, Alex; SADDIK, A. E. Design and Development of a Cloud Based Cyber-Physical Architecture for the Internet-of-Things. In: *2015 IEEE International Symposium on Multimedia (ISM).* IEEE, 2015. str. 459-464

[12] Cisco, "TCP/IP Overview", Avgust 2005. [Online]. Dosegljivo: http://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13769-5.html. [Dostopano 31. 3. 2016].

[13] P. Duffy, Cisco, "Beyond MQTT: A Cisco View on IoT Protocols", April 2013. [Online]. Dosegljivo:

http://blogs.cisco.com/digital/beyond-mqtt-a-cisco-view-on-iot-protocols. [Dostopano 31. 3. 2016].

[14] Micrium, "IoT for Embedded Systems: The New Industrial Revolution Part 3: IoT protocol stack options". [Online]. Dosegljivo: https://www.micrium.com/iot/internet-protocols/. [Dostopano 31. 3. 2016].

[15] Embedded, "UDP and the embedded wireless Internet of Things". [Online]. Dosegljivo: http://www.embedded.com/electronics-blogs/cole-bin/4229531/UDP—the-embedded-wireless–Internet-of-Things-. [Dostopano 31. 3. 2016].

[16] PHP: Hypertext preprocessor, "What is PHP?". [Online]. Dosegljivo: http://php.net/manual/en/intro-whatis.php. [Dostopano 31. 3. 2016].

[17] phpMyAdmin, "About". [Online]. Dosegljivo: https://www.phpmyadmin.net/. [Dostopano 31. 3. 2016].

[18] DigitalOcean, "How To Use Apache HTTP Server As Reverse-Proxy Using mod_proxy Extension". [Online]. Dosegljivo: https://www.digitalocean.com/community/tutorials/how-to-use-apache-http-server-as-reverse-proxy-using-mod_proxy-extension. [Dostopano 31. 3. 2016].

[19] GNU General Public Licence. [Online]. Dosegljivo: https://www.gnu.org/copyleft/gpl.html. [Dostopano 20. 9. 2014].